CodeWizards 2016

Rules

Revision 1.2.1

November — December, 2016

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 2016 to 24.00 hours 25 December 2016 Moscow time.

First week (from 00.00 hours on 7 November 2016 to 24.00 hours on 13 November 2016) and fourth week (from 00.00 hours on November 28, 2016 to 24.00 hours 4 December 2016) 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 26 November 2016 to 24 hours 00 minutes 27 November 2016;
- the second stage — from 00 hours 00 minutes on 10 December 2016 to 24 hours 00 minutes 11 December 2016;
- the third stage (final) — from 00 hours 00 minutes 17 December 2016 to 24 hours 00 minutes 18 December 2016.

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 2016 to 24.00 hours on 25 December 2016 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 the documentation in Chapter 2.

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 13";
- 2nd place — Apple Macbook Air 13";
- 3rd place — Apple iPad;
- 4-6 places — prizes;
- 1-6 places in the Sandbox — prizes.

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 the world of CodeWizards 2016

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

This competition gives the opportunity to test your programming skills by creating artificial intelligence (strategy) to control a wizard in a special game world (read more about the features of the world of CodeWizards 2016 in following sections).

Rules of the competition are based on the MOBA genre, which is popular in the world of computer games. In each game you will face five strategies of others players. Also you will have four allies. Five strategies located on the same side are faction: Academy or Renegades. The main goal of these five players is to destruct the base of the opposing faction. The main personal goal of every wizard is to collect the highest possible number of score points. The title of winner of the game, as well as all the other places are distributed in accordance with the number of score points. Two or more players can share one place if their scores are equal. The player gets points if his wizard damages, destroys or is just near in the time of death a of a unit of another faction. Also the player gets points for some other actions. All players of the faction get a significant number of points in case of reaching the main team goal.

The rules of the game are nearly identical to the classical canons of the genre. Bases of factions are connected by three paths (top, middle and bottom). Between these paths there are forests. The guardian towers are situated on the paths: 2 towers of each faction on each path. Thus, in the beginning of the game there are 14 buildings on the map. With a certain time bases of each faction generate 3 similar squads of wizards' minions: one for each path. They immediately run on their path in the direction of the opposing faction base, attacking all enemies on their way.

The tournament is held in several stages, preceded by a qualification in the Sandbox. Sandbox is a competition, which takes place during all period of the competition. On each stage the player has a certain rating value. It is an indicator of how well his strategy is playing in the games.

The initial value of the rating in the Sandbox is 1200. At the end of any game this value can both increase and decrease. The victory over the weak (low ranking) opponent gives a small increase, and the defeat of a strong opponent slightly reduces your rating. Over the time, the rating in the Sandbox becomes more inert, thereby the influence of random long series of wins and defeats by the place of the participant reduces. However, it makes difficult to change the participant's position with a substantial improvement of his strategy. To cancel this effect, the participant may reset the variability of the rating to the starting value while submitting new strategy (for this action it is necessary to activate the appropriate option). If the new strategy is accepted by the system, participant's rating will be reduced by a significant amount after next game in the Sandbox. But further participation in games will recover the rating quickly, and it will even become higher, if your strategy really became more effective. It is not recommended to use this option for minor, incremental improvements of your strategy, as well as in cases when a new strategy is insufficiently tested and the effect of changes in it is not known.

The initial value of the rating at each main stage of the tournament is 0. For each game participant gets a certain number of rating units, which depend on his place (a system is similar to the system, which is used in "The FIA Formula One World Championship"). If two or more participants share same place, the total number of rating units for this place and for the following `number_of_such_participants` − 1 places is divided equally between these participants. For example, if two participants share the third place, each receive half of the sum of rating units for third and fourth places. The result of the dividing is always rounded down. Detailed information about the stages of the tournament will be provided in announcements on the project website.

Initially, all participants can only take part in Sandbox games. Players can send their strategies to the Sandbox, and the last strategy accepted will be used by the system for qualification games. Each player takes part in about one qualification game an hour. The jury can change this interval based on the capabilities of the testing system; however, for most participants it will remain constant. There are a number of criteria that warrant the interval between qualification games to be increased for a specific player. For each week past since the moment of last accepted submission the participation interval for a player is doubled. For each "crashed" strategy in 10 most recent Sandbox games, an additional fine is accrued, which equals to 30% of the base testing interval. For more detail on the reasons why a strategy can be «crashed» you can refer to subsequent sections.

Games in the Sandbox follow a set of rules that matches the rules of a random finished tournament stage or the rules for the next (current) stage. The closer two players' ratings are to each other in the Sandbox, the more likely they are to be in the same game. The Sandbox starts before the first stage of the tournament and ends some time after the final stage (see the Schedule of stages for detail). Additionally, the Sandbox is frozen while tournament stages are in progress. The results of games in the Sandbox are used to select players for Round 1, where 1080 participants with the highest ratings are selected (if players have equal ratings, the priority is given to the player who sends their final strategy version earlier), as well as additional players for subsequent stages of the tournament, including the Finals.

Tournament stages:

- In **Round** 1 you will learn the rules of the game and basic controls of a Wizard. Each game in this stage will consist of 10 players, who will be distributed into two factions, so that the difference between current ratings[1] of the participants in the two factions is minimum. The Wizard with the highest rating in each faction is appointed the Supreme Wizard. His strategy can send messages to other wizards from the same factions; it is also provided more computing resources (more CPU time). Functionality of the messages is limited, only allowing to send[2] other Wizards to a certain lane. In this mode, the Wizards can only use staff hit and «Magic Missile» spell, while the living energy of all structures is limited to half of the normal level. Damage ratio on accidental hit of a friendly Wizard equals 25%. Regardless of the stage of the championship friendly minions and structures take 0% of damage. Round 1, like all subsequent stages, consists of two parts, with a short break between them (the Sandbox is resumed during the break), which allows the players to improve their strategies. For games in each part, the latest strategy sent by a player before the beginning of that part is chosen by the system. Games are conducted in waves. In each wave, each player takes part in exactly one game. The number of waves in each part is determined by the testing system capacity, but it is guaranteed to be at least ten. 300 participants with the highest ratings make it to Round 2. For Round 2, 60 participants with the highest ratings in the Sandbox (as of the beginning of Round 2), chosen from among those who didn't make it to Round 1.
- In **Round** 2 you will be improving your Wizard control skills and study the mechanics of the Wizard reaching new levels and learning new skills. The right approach to selecting and using skills is key to winning this stage. The way players are selected for the games is similar to Round 1, but the Supreme Wizard has more skills this time. He can tell what skills other wizards in the factions should learn. Structures in this round have normal amount of living energy, and damage ratio on accidental hits of a friendly wizard equals 50%. To additionally complicate the task, after Round 1 results are summarized, some of the weaker strategies will be discarded, so you will have to play against stronger opponents. At the end of Round 2, the best 50 strategies will make it to the Finals. 10 participants will additionally be selected

[1] When a game is conducted as part of Round 1, the ratings in this round are considered. When a game is conducted in the Sandbox using Round 1 rules, the Sandbox ratings are considered.

[2] Regardless of the tournament stage, messages from the Supreme Wizard can be partially or fully ignored by other Wizards' strategies.

to the Finals, with the highest ratings in the Sandbox (as of the Finals starting), from among those who didn't make it to the main tournament.

- **The Finals** is the most important stage in the tournament. After the first two stages, only the strongest players will remain. So in each game, you will be up against the strongest. Exactly. To manage five wizards in your faction, 5 copies of your strategy will be running. Five wizards in the opposing faction will be controlled by 5 copies of the opponent's strategy. A random wizard is appointed the Supreme Wizard in each faction. He can control other wizards in that faction by sending messages in binary format. The restriction on binary data size is relatively high, but the message can only be received after a delay, which is proportional to its length. Damage ratio on accidental hit of a friendly wizard equals 100%. The winner is identified by adding up the points scored by all wizards in each faction. Otherwise the rules of the game are the same as in Round 2. The Finals do have their peculiar features, however. The stage is still divided into two parts, but these no longer consist of waves. Each half of the Finals features games between all pairs of participants. The operation will be repeated if the time and capacity of the testing system allow.

After the Finals, all finalists are ranked according to their scores in descending order. In case of equal ratings, the participant whose strategy for the Finals was sent earlier is ranked higher. Prizes for the Finals are awarded based on the place in the final ranking. Top six finalists are awarded prizes:

- 1st place — Apple Macbook Pro 13";
- 2nd place — Apple Macbook Air 13";
- 3rd place — Apple iPad;
- 4-6 places — valuable gifts.

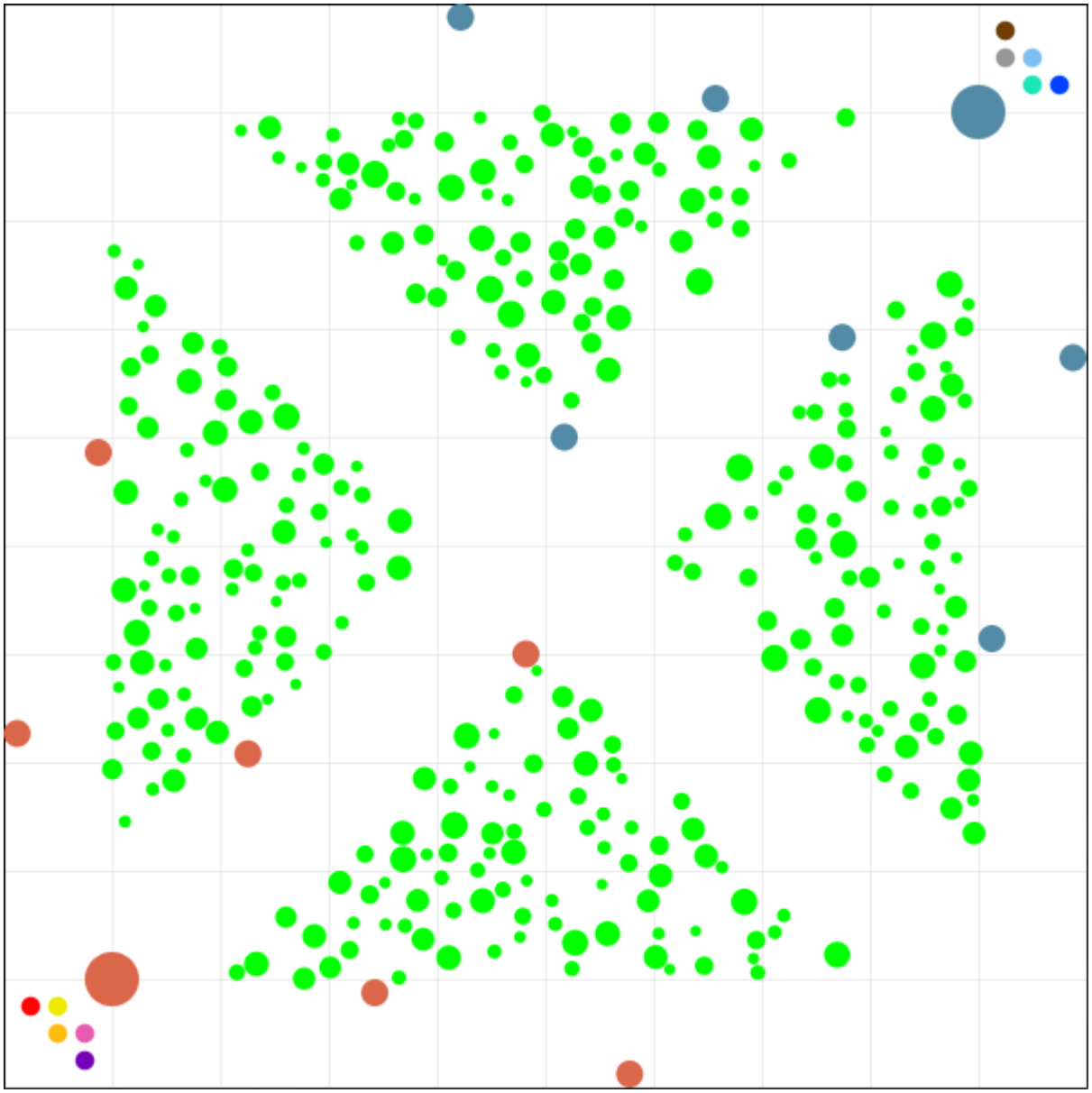
After the Sandbox is over, all its participants, except for the Finals prize winners, are ranked according to their scores in descending order. In case of equal ratings, the participant who sent the latest version of his/her earlier is ranked higher. Prizes for the Sandbox are awarded based on the place in the final ranking. Top six players in the Sandbox are awarded valuable prizes.

2.2 Description of the game world

The game world is two-dimensional, and all units are round. The horizontal axis in this world is directed from left to right and vertical axis from top to bottom; the direction 0.0 matches the horizontal axis direction, and positive rotation angle means clockwise rotation. Playing area is a square with the upper left corner at (0.0, 0.0), where each side length equals 4000.0. No living unit can leave the playing area.

In the map layout below, copper color indicates the Academy buildings (larger circle means the faction base, smaller circles indicate guardian towers), blue steel color indicates the Renegades' structures, green means trees. 10 color sports (5 in the lower left corner of the map for Academy and 5 in the upper right corner of the map for Renegades) indicate the Wizards' spawning positions. It should be noted that coordinates of the circle centers and tree radiuses in the forests can be different between games. New trees can also appear in the course of the game.

The game engine transforms coordinates and directions before send it to the strategy of renegade wizard. So the strategy always "think" that it plays for the bottom left side, and the enemy is in the top right corner.



The time in the game is discrete, measured in «ticks». At the beginning of each tick the game receives from the strategies the wizards' desired actions in this tick and updates the status of each wizard in accordance with these desires and the limitations of the game world. Next the game calculates changes in the world and the objects it contains, which took place during the tick, and the process is repeated with the updated data. Maximum duration of any game equals 20000 ticks, but the game can be terminated early if either faction has reached its team goal, or all participants strategies have been «crashed». It is extremely unlikely, but not impossible, that both factions will reach their team goals in the same tick. Then additional points are awarded to all game participants.

«Crashed» strategy is not able to control a wizard anymore. The strategy is considered «crashed» in following cases:

- The process of the strategy unexpectedly exited, or there was a problem with communication protocol between the strategy and a game server.
- The strategy exceed one of the time limits. For one tick the strategy has 10 seconds of wall time. For all the game the strategy process usually has

  $$20 \times <game\_duration\_in\_ticks> + 10000 \quad (2.1)$$

  milliseconds of wall time and

  $$10 \times <game\_duration\_in\_ticks> + 10000 \quad (2.2)$$

  milliseconds of processor time. The master wizard strategy has

  $$30 \times <game\_duration\_in\_ticks> + 10000 \quad (2.3)$$

  milliseconds of wall time and

  $$20 \times <game\_duration\_in\_ticks> + 10000 \quad (2.4)$$

  milliseconds of processor time.[3] Time formula includes max game duration. This limit covers as a user code time as a client-server communication time. The time limit remain a constant even if the real game time is different.
- The strategy exceeded a memory limit. Strategy process can not use more than 256 MB of RAM at the same time.

Discovering units on the map is limited by the fog of war. The strategy takes the information only about the units, locating in vision range [4] of the wizard or any unit from his fraction.

2.3 Unit classes

There are 6 classes of units in the CodeWizards 2016 world, some of which are then divided into types:

- wizards;
- projectiles: Magic Missile (`MAGIC_MISSILE`), Frost Bolt (`FROST_BOLT`), Fireball (`FIREBALL`) and Dart (`DART`);
- bonuses: Empower (`EMPOWER`), Haste (`HASTE`) and Shield (`SHIELD`);
- structures: Faction Base (`FACTION_BASE`) and Guardian Tower (`GUARDIAN_TOWER`);
- minions: Orc-woodcutter (`ORC_WOODCUTTER`) and Fetish Blowdart (`FETISH_BLOWDART`);
- trees.

[3] Despite the limit for the wall time is notably higher, than limit for the processor time, artificial «slowdowning» of strategy with commands like «`sleep`» is prohibited (any other attempts to slowdown/destabilize the system are prohibited too. In case of such behavior, the jury can disqualify user from the competition and block his account.

[4] The distance between the units is the distance between their geometric centers.

Wizards, structures, minions and trees are living units. The main characteristics of every living unit are their current and maximum amount of living energy. Generally, when living energy drops to zero, a unit is considered dead and is removed from the game world. Wizards are the only living units that feature health regeneration. Each tick, they automatically recover some of their living energy. Regeneration rate is a real number, usually less than one. Over several ticks it may look like a wizard is not regenerating health, but that is not true. Total regeneration over several ticks is stored in a special pool. A wizard is considered dead if the integer part of his living energy drops to zero, regardless of how much energy is currently stored in the pool.

Comparative characteristics of the living units are shown in the table below:

Living unit characteristics	Wizard	Orc	Fetish	Guardian Tower	Faction Base	Tree
Radius	35	25	25	50	100	20 – 50
Max life	100[5]	100	100	500[6]	1000[6]	6 – 30
Vision range	600	400	400	600	800	—
Ranged attack distance[7]	500	—	300	600	800	—
Melee attack distance[8]	70	50	—	—	—	—
Attack cooldown	30[9]	60	30	240	240	—
Attack damage	12[10]	12	6	36	48	—

A bonus has a radius of 20, regardless of the bonus type.

Comparative characteristics of the projectiles are shown in the table below:

Projectile characteristics	Magic Missile	Frost Bolt	Fireball	Dart
Radius	10	15	20	5
Speed	40	35	30	50
Damage on direct hit	12	24	24[11]	6

In addition to causing damage, the frost bolt also freezes (`FROZEN`) the target for 60 ticks. The frozen unit cannot move or perform any actions, but the number of ticks remaining until the next action can be applied keeps counting down. Buildings and trees cannot be frozen. A wizard's strategy does not gain control until the freezing period is over.

The fireball explodes when it hits the target or reaches maximum flight distance, causing damage to all living units within the distance of 100 between the center of the fireball and the closest point of the unit. All units receiving damage from the explosion also get the burning status (`BURNING`). It remains in effect for 240 ticks, causing 24 units of damage over this time. Damage from several statuses of the same type is accumulated.

A building attack does not create a projectile. Damage is caused instantly due to a magic blast.

The second Guardian Tower on the lane is immune to damage if the first tower is alive. To be able to attack the opposite Faction Base the wizards should destroy both enemy towers on any lane.

---

[5] Maximum living energy for a zero-level wizard. Maximum energy is increased by 10 for each level achieved.

[6] Actual living energy of a structure at a specific stage of the tournament can be different from the value shown in the table.

[7] Maximum duration of a projectile flight or maximum distance between unit centers.

[8] Maximum possible distance to the nearest point of the target.

[9] Minimum possible interval between a wizard's two consecutive attacks or spells. Each action type has its own limited interval.

[10] Base damage from a close-combat attack or the simplest spell, «Magic Missile». The damage can be increased by the wizard learning certain skills.

[11] Formally, a fireball does not inflict any damage on direct hit, but as it explodes on impact, it causes damage to all living units within the distance of 100 between the center of the fireball and the nearest point of the living unit, including the unit that was directly hit. If said distance does not exceed the fireball radius, the explosion causes maximum damage. As distance increases, the damage is reduced in a linear manner, until it reaches half of the value in the table at the periphery of the explosion radius.

2.4 Wizard characteristics

The main characteristics of every living unit are current and maximum amount of living energy. However, a wizard also has several other important characteristics. For example, the wizard's ability to cast spells is limited by current and maximum amount of magic energy and its regeneration rate.

Unlike other living units, a wizard cannot be killed; you can only destroy his physical shell. After some time the wizard will respawn in a new body. Respawn can occur at least 1200 ticks after the wizard's death and at least 2400 ticks since his last respawn. The wizard respawns at his respawn position, or next to it if the position is currently occupied.

In some game modes, the wizard can accumulate experience and increase its level. Every wizard starts with zero level and grows to the maximum level of 25. Nevertheless, the rules of the game are balanced so that achieving level 15 is almost impossible. The wizard gets experience in the same amount and on the same occasions as when the player controlling him gets points, unless the wizard is dead and awaits respawn at that tick. A wizard needs to score 50 experience points to get to level one; an additional 100 points for level two, 150 for level three, 200 for level four, and so on.

The amount of living energy at level zero equals the amount of magic energy and equals 100 points. Both types of energy grow 10 points each level. Living energy regenerates at 0.05 ticks^{-1} at level zero, and increases by 0.005 ticks^{-1} with each level. The respective characteristics of magic energy are 4 times higher.

A wizard can learn exactly one skill with each level achieved. Skills are divided into 5 groups. The skills within each group are ordered and can be learned only in sequence. The number of skills in each group equals 5. The first and the third skill in each group, as a rule, increase one of the wizard's characteristics. The second and fourth one are auras. An aura operates similarly to a passive skill, but is applied not just to the wizard himself, but also to all friendly wizards within the distance of 500 or less. If several auras apply to a wizard that improve one and the same quality of the unit, only the one with the biggest effect is taken into account. The final, fifth skill in each group is the most important («ultimate») skill. It enables the wizard to cast a new spell or substantially improves an existing spell.

- Passive skills and auras in the first group increase the distance of the wizard's spells by 25 for each level. Thus, a wizard who has mastered all these skills will be able to cast spells at the distance of 600. Additionally, all friendly wizards nearby will get extra distance of 50 for their spells. The last skill in this group allows using the «Magic Missile» spell without a delay. By default, the delay in casting this spell is 60 ticks. Nevertheless, the overall delay on a wizard's actions still applies.
- Passive skills and auras in the second group increase the damage caused by magic projectiles by 1 for each level. The last skill in this group allows the wizard to use the «Frost Bolt» spell, which causes substantial damage to a single opponent, in addition to freezing him for 60 ticks.
- Passive skills and auras in the third group increase the damage caused by staff hit in close combat by 3 for each level. The last skill in this group allows the wizard to cast the «Fireball» spell, which causes damage to a group of opponents, in addition to setting them on fire.
- Passive skills and auras in the fourth group increase the wizard's maximum possible movement distance in one tick by 5% for each level. The total increase from the four skills is 20%, while aura adds 10% movement to friendly wizards nearby. The last skill in this group enables the wizard to use the «Haste» spell, which gives a friendly wizard the `HASTENED` status for 1200 ticks. This status adds an extra 30% to the unit's movement speed, in addition to increasing the turning speed by 50%. Several statuses of the same type are not accumulated.
- Passive skills and auras in the fifth group decrease the damage received from magic projectiles and structure attacks by 1 for each level. The last skill in this group allows the wizard to cast the «Shield» spell, which gives a friendly wizard the `SHIELDED` status for 1200 ticks. This status reduces any direct damage by 33.3%. The only exception is the damage from `BURNING` status. Several statuses of the same type are not accumulated.

You can find the full list of available skills in the documentation to enumeration `SkillType` (Chapter 4).

2.5 Control of the wizard

In the beginning of each tick, the game simulator sends data about the current state of the part of the world, visible to the wizard, to the strategies of each alive wizard. In response the strategy sends a set of instructions for control of the wizard, encapsulated in the object of the `Move` class.

These instructions are processed in the following order:

- Each wizard learns the skill `move.skillToLearn`, if it is set. The game simulator will ignore the instruction, if the number of skills learned by the wizard is already equal to his level or if the wizard has not yet learned all the previous skills in the group.
- Then wizards randomly perform actions, set in `move.action`:
  - `NONE`. *Don't do anything.*
  - `STAFF`. *Strike with the staff.* The attack will strike all living units which centers are in the sector of $-\pi/12$ to $\pi/12$. The distance from the center of the wizard to the nearest target point must not exceed 70. If the wizard died in the result of the staff strike of another wizard, made earlier within the same tick, then the staff strike will still be performed. The dead wizard will not gain experience for the caused damage, nevertheless, the player whose strategy controls this wizard, will receive a relevant amount of points.
  - `MAGIC_MISSILE`, `FROST_BOLT` or `FIREBALL`. *Create the relevant magic projectile.* When creating a projectile, its center is set equal to the center of the wizard, and its direction is determined as `wizard.angle + move.castAngle`. Value `move.castAngle` is set by the strategy and is limited by the interval from $-\pi/12$ to $\pi/12$. Additional parameters `move.minCastDistance` and `move.maxCastDistance` determine the minimum and the maximum distance of the projectile cast. If the distance from the center of the projectile to the point of its appearance is less than `move.minCastDistance`, then the projectile will pass through all other game objects, except for trees. If the distance from the center of the projectile to the point of its appearance is more than `move.maxCastDistance`, then the missile will be removed from the game world. At that moment, the projectile of the `FIREBALL` type will explode. Collisions of the magic projectile with the wizard, who created it, are ignored. If the wizard died in the result of the staff strike of another wizard, made earlier in the same tick, then the missile will not be created.
  - `HASTE` or `SHIELD`. *Apply relevant magic status to the target.* The status is applied to a friendly wizard with identifier `move.statusTargetId` or to the wizard himself if such a wizard is not found. The target must be located in the sector from $-\pi/12$ to $\pi/12$, and the distance to it must not exceed `wizard.castRange`. Value `wizard.castRange` for a wizard of the zero level is equal to 500, but it may grow to 600 after some skills are learned. If the wizard died in the result of the staff strike of another wizard, made earlier within the same tick, the status will not be applied. The wizard that casts a status spell on any friendly wizard automatically gets the same status.

  There must be not less than 30 ticks between any two consecutive acts of the wizard, different from `NONE`. For each act there is its own delay, which limits consecutive use of two actions of one type. Creation of a missile or application of the status consumes magic energy of the wizard. Desired action will be ignored if there is not enough energy.

Characteristics of the act	Staff strike	Acceleration	Shield
Cooldown	60	120	120
Manacost	—	48	48

Characteristics of the act	Magic Missile	Frost Bolt	Fireball
Cooldown	60[12]	90	120
Manacost	12	36	48

- Then wizards are again randomly put in order and they are moved.

  In the CodeWizards world, 2016 there is no inertia, and the change of speed happens instantaneously. Strategy request `move.speed` determines the move of the wizard forward/back, and `move.strafeSpeed` — sideways movement. Value `move.speed` is limited by the interval from -3.0 to 4.0, where positive values mean the forward movement, and negative — backward movement. Value `move.strafeSpeed` is limited by the interval from -3.0 to 3.0, where positive values mean the right side movement, and negative values — left side movement. Limits of both intervals may be extended depending on the skills learned by the wizard, depending on the influence of some auras, as well as in case of application of the `HASTENED` status. Both types of movement may be performed simultaneously within one tick, but in that case an additional limitation will be applied: if value $\sqrt{(\frac{move.speed}{maxSpeed})^2 + (\frac{move.strafeSpeed}{maxStrafeSpeed})^2}$ is more than 1.0, then both settings of the movement speed (`move.speed` and `move.strafeSpeed`) will be divided by this value by the game simulator. `maxSpeed` and `maxStrafeSpeed` — relevant limitations, applied to the movement of the wizard this tick (with consideration of the direction of the movement, learned skills and affecting auras and statuses).

  Movement of wizards is performed sequentially, according to the selected order. At that, partial movement of the wizard is not applied. If it is impossible to change the location of the wizard for the whole value of the movement, set by the strategy[13], then his movement is postponed. After the iteration is completed, the game simulator again iterates all the wizards and attempts to relocate those, whose position has not yet changed within this tick, This happens until all the wizards have been relocated. If in the course of the next iteration none of the wizards was moved, then the operation will be terminated.

- All wizards turn for the angle set by the strategy. By default the absolute angle change can not be higher than $\pi/30$. Hastened wizard turns 1.5 times faster.

Apart from the listed actions, the supreme wizard can send messages to other wizards from his faction. Field `message.lane` contains a recommendation to go to the relevant path. Field `message.skillToLearn` becomes available in Round 2 and contains a recommendation to learn the specified skills. A skill may require learning of other preliminary skills or may be unavailable due to the low level of the wizard. The wizard is recommended to memorize the task and move towards its implementation. At that, a later task has a higher priority. In the Finals the supreme wizard has the opportunity to send text messages. Field `message.rawMessage` is a byte array, its maximum length is 1024. Usually wizards receive messages in the next tick after the dispatch, but if there is a text part, the receipt will be delayed for `ceil(message.rawMessage.length / 2.0)` game ticks. If there is a registered message in the system for one of the wizards, but it has not yet been received by him, then new messages to this wizard will be ignored.

2.6 Other game objects

Each 750 ticks the base of each faction generates 3 troops of minions: per one for each of the lanes. Each troop consists of three orks and one fetish. The troop directly rushes down its path in the direction of the base of the opposite faction, attacking all enemies on its way. Wizards use minions as the cannon fodder. At that, they are trying to stay in the safe area and attack the enemy from the distance.

In the forest area neutral minions may appear with some probability. Usually they are not aggressive, but if one of the minions is damaged, all the neutral minions in the area will rush to the offender, attacking anyone on their way.

A minion can move strictly forward not faster than 3 ticks^{-1}. The absolute angle change of a minion can not be higher than $\pi/30$.

Every 2500 ticks up to two bonuses may appear on the map. The bonuses are created at the two following points:

[12] Delay in the application of the spell «Magic Missile» may be less in the result of learning of some skills.

[13] After the relocation a wizard is partially or completely beyond the limits of the map or crosses some other alive unit.

(1200, 1200) and (2800, 2800). If any part of the new bonus area is already occupied by a wizard or by an existing bonus, then the creation of the bonus will be postponed till the end of the next interval.

2.7 Collision of units

- Collision of alive units between them as well as with the borders of the map is not allowed by the game simulator.
- If the distance from the center of an alive unit to the center of the projectile is less than or equal to the sum of their radii, then the alive unit will be damaged, and the projectile will be removed from the game world. At that, the fireball explodes and damages all alive units in the area.
- If the distance from the center of a wizard to the center of the bonus is less than or equal to the sum of their radii, then the wizard receives a magic status for 2400 ticks, depending on the bonus type:
  - `EMPOWER` increases the damage caused by a wizard in the result of the staff strike and in case of hitting the target with the magic projectile by 1.5 times.
  - `HASTE` speeds up the movement of a wizard similar to the cognominal spell.
  - `SHIELD` decreases the damage, received by a wizard, similar to the cognominal spell.

All types of collisions not described in this document are ignored by the game simulator.

2.8 Scoring

The wizard receives experience in the same amount and in the same cases, when the player, controlling the wizard, receives points, except for when the wizard is dead in the tick when points are scored and is waiting for respawn. Points are given for the following actions:

- Causing damage to buildings and wizards of the opposing faction. Damages are accounted with coefficients of 0.5 and 0.25 respectively. Rounding is done down to the nearest integer. Points for damages are given only to the player, whose wizard caused these damages.
- Destruction of buildings or wizards of the opposing faction, as well as minions of any faction, different from the wizard's faction. Total reward for buildings is 50% of the maximum amount of their life energy, for minions is 25%, for wizards is 100%. Destruction may be done by the wizard himself or by any other unit from his faction. Experience is equally distributed among all friendly wizards, the distance to whom from the target does not exceed 600. At that, if the number of such wizards is more than one, then the total reward increases by 67%. Rounding is done down to the nearest integer.
- Collection of any bonus gives 150 score/experience points.
- In case of destruction of the opposing faction base, all friendly players receive 1000 points. The game is finished in this case.

Глава 3

Creation of the strategy

3.1 Technical part

To create a strategy, first you need to do is to choose one of the supported programming languages[14]: 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). Possibly this set will be extended. On the project website you can download the user pack for each of the languages. It is allowed to modify only one file in the pack, which is meant for maintenance of your strategy, for example MyStrategy.java (for Java) or MyStrategy.py (for Python)[15]. All other files of the pack in the course of compilation of the strategy will be replaced with standard versions. Nevertheless, you can add your own files with code to the strategy. These files must be placed in the same directory, as the main strategy file. When submitting the solution, all of them must be placed into one ZIP-archive (all of them must be located in the archive root). If you do not add new files in the pack, it is enough to send the strategy file (via the dialog when selecting the file) or input its code into the text field.

After you sent your strategy, it is put in queue for testing. First, the system will try to compile a submission with your files, and then in case of success, it will create several short (of 200 ticks) games of different formats[16]: 10×1, 10×1 with unlocked skills of wizards and 2×5. In order to control the wizard of each of the participants of these games, a separate process with your strategy will be launched, and for this strategy to be considered accepted (correct), none of the copies of the strategy may «crash». The players in the test games will be assigned names in the format «`<name of_player>`», «`<name of_player> (2)`», «`<name of_player> (3)`» and etc.

After the successful completion of the described process, your submission receives the status «Accepted». The first successful submission means also your your registration in the Sandbox. You are assigned the start rating (1200), and your strategy starts participating in periodic qualification games (see the description of the Sandbox for more details). Also the function to create your own games becomes available, in which you can choose the strategy of any player as opponent (including your own), created before the moment of your last successful submission. The games created by you do not affect your rating.

---

[14] For all programming languages 32-bit compiler/evaluator versions are used.

[15] C++ is the exception, because for it, it is possible to modify two files: MyStrategy.cpp and MyStrategy.h. At that, the presence in the archive of the MyStrategy.cpp file is mandatory (otherwise the strategy cannot be compiled), and the presence of the MyStrategy.h file — optional. In case of its absence a standard file from the pack will be used.

[16] Main parameters of the game format are the number of players participating in the game, and the number of units under control of each player. In a short form, the format is written as `<number_players>` × `<number_units>`, for example, the string 4×3 means that the game has a format with 4 players, and each controls three units. The format may be complimented with the clarification, if the short notation of formats for different stages of the championship is the same. Pay attention, that in the Final games of this championship, each wizard of one faction is controlled by the participant's strategy launched as a separate process. From the point of view of the game API, these wizards will be virtually distributed among five different players, but determination of the winner will only depend on the sum of their points, and not on distribution to specific wizards.

The system has limitations in relation to the number of submissions and user games, specifically:

- It is forbidden to send strategies more often than three times per twenty minutes.
- You cannot create more than three user games in twenty minutes.

In order to simplify the adjustment of small changes in the strategy, the system has the possibility to perform a test submission (flag «Test submission» in the strategy submission form). The test submission will not be displayed to other users, does not participate in the qualification games in the Sandbox and games of the tournament stages, also it is impossible to create games with it by oneself. Nevertheless, after the system accepts this submission, the system will automatically add the test game with ten participants (format10×1): the test submission itself and nine strategies from the «Quick start» section. The test game will be visible only to the participant, who created this test submission. The base duration of this test game is 2000 ticks. The dispatch of test games is limited in the same way as the dispatch of normal submissions. Test games do not influence the frequency of games creation by the user.

The players have a possibility to view past games in the special visualizer. To do that it is necessary to press the «View» button in the list of games or press the «View game» button on the page of the game.

If you watch the game with participation of your strategy and noticed something strange in its behavior, or your strategy does not do what you expect from it, you can use a special utility Repeater to perform a local repeat of the game. The local repeat of the game — is the opportunity to launch the strategy on your computer in such a way, that it sees the game world around the same way, as it was on the server while testing. This will help you to perform adjustments, add logging and watch the reaction of your strategy during each game moment. To do that, download Repeater from the CodeWizards 2016 website (section «Documents» → «Repeater Utility») and unzip it. To launch Repeater, you need installed software Java 8+ Runtime Environment. Pay attention, that any cooperation of your strategy with the game world during the local repeat is completely ignored. This means, that at each moment of time the surrounding world looks exactly the same for the strategy, as it looked during the game in the course of testing on the server, and does not depend on the actions your strategy performs. The Repeater utility only posses the data, which were sent by your strategy, but does not posses the complete recording of the game. Thus the visualization of the game is not possible. You can read more information about the Repeater utility in the relevant section on the website.

Apart from everything mentioned above, the players have an opportunity to launch simple test games locally on their computer. To do that, it is necessary to download a zip-file with the Local runner utility from the website section «Documents» → «Local runner». Use of this utility will allow you to test your strategy in conditions, similar to the conditions of the test game on the website, but without any limitations in relation to the number of created games. Renderer for local games is significantly different from the renderer on the site. All game objects in it are displayed schematically (without colorful models). It is very easy to create a local test game: launch Local runner with the help of the relevant launch script (*.bat for Windows or *.sh for *nix systems), then launch your strategy from the development area (or any way that is convenient for you) and watch the game. During local games, you can make adjustments to your strategy, put break points. But remember, that Local runner waits for the strategy response for not more than 30 minutes. Upon expiry of this time, it will consider the strategy «crashed» and will continue to work without it.

3.2 Control of the wizard

In the beginning of the game an object of the `MyStrategy` class will be created for your wizard, and in the field of this object the strategy can save information about the course of the game. Control of the wizard is performed via the `move` method of the strategy, which is called once per tick. The method receives the following parameters:

- the wizard `self`, for which the method is called;
- current state of the world `world`;
- set of game constants `game`;
- object `move`, by setting the parameters of which the strategy determines the behavior of the wizard.

Implementation of the user shell of the strategy in different languages may differ, nevertheless, in the general case it is **not** guaranteed, that in case of different calls of the method `move` it will receive pointers to the same objects as the parameters. Thus, it is not allowed, for example, to save pointers to objects `world` or `player` and in the course of the next ticks receive updated information about these objects by reading their fields.

3.3 Implementation examples

Hereinafter there are simple examples of strategies for all programming languages, these strategies move the wizard forward and to the right, at the same time turning him to the right. Additionally strategy requests to cast «Magic Missile». You can find complete documents according to classes and methods of Java in the following chapters.

3.3.1 Java example

```
import model.*;

public final class MyStrategy implements Strategy {
    @Override
    public void move(Wizard self, World world, Game game, Move move) {
        move.setSpeed(game.getWizardForwardSpeed());
        move.setStrafeSpeed(game.getWizardStrafeSpeed());
        move.setTurn(game.getWizardMaxTurnAngle());
        move.setAction(ActionType.MAGIC_MISSILE);
    }
}
```

3.3.2 C# example

```
using Com.CodeGame.CodeWizards2016.DevKit.CSharpCgdk.Model;

namespace Com.CodeGame.CodeWizards2016.DevKit.CSharpCgdk {
    public sealed class MyStrategy : IStrategy {
        public void Move(Wizard self, World world, Game game, Move move) {
            move.Speed = game.WizardForwardSpeed;
            move.StrafeSpeed = game.WizardStrafeSpeed;
            move.Turn = game.WizardMaxTurnAngle;
            move.Action = ActionType.MagicMissile;
        }
    }
}
```

3.3.3 C++ example

```
#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 Wizard& self, const World& world, const Game& game, Move& move) {
    move.setSpeed(game.getWizardForwardSpeed());
    move.setStrafeSpeed(game.getWizardStrafeSpeed());
    move.setTurn(game.getWizardMaxTurnAngle());
    move.setAction(MAGIC_MISSILE);
}

MyStrategy::MyStrategy() { }
```

3.3.4 Python 2 example

In Python 2 current wizard's name is changed from «`self`» to «`me`».

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


class MyStrategy:
    def move(self, me, world, game, move):
        """
        @type me: Wizard
        @type world: World
        @type game: Game
        @type move: Move
        """
        move.speed = game.wizard_forward_speed
        move.strafe_speed = game.wizard_strafe_speed
        move.turn = game.wizard_max_turn_angle
        move.action = ActionType.MAGIC_MISSILE
```

3.3.5 Python 3 example

In Python 3 current wizard's name is changed from «`self`» to «`me`».

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


class MyStrategy:
    def move(self, me: Wizard, world: World, game: Game, move: Move):
        move.speed = game.wizard_forward_speed
        move.strafe_speed = game.wizard_strafe_speed
        move.turn = game.wizard_max_turn_angle
        move.action = ActionType.MAGIC_MISSILE
```

3.3.6 Pascal example

In Pascal current wizard's name is changed from «`self`» to «`me`».

```
unit MyStrategy;

interface

uses
  StrategyControl, TypeControl, ActionTypeControl, BonusControl, BonusTypeControl, BuildingControl,
  CircularUnitControl, FactionControl, GameControl, LaneTypeControl, LivingUnitControl, MessageContr
  MinionTypeControl, MoveControl, PlayerContextControl, PlayerControl, ProjectileControl, Projectile
  SkillTypeControl, StatusControl, StatusTypeControl, TreeControl, UnitControl, WizardControl, World

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

  end;

implementation

uses
  Math;

procedure TMyStrategy.Move(me: TWizard; world: TWorld; game: TGame; move: TMove);
begin
  move.Speed := game.WizardForwardSpeed;
  move.StrafeSpeed := game.WizardStrafeSpeed;
  move.Turn := game.WizardMaxTurnAngle;
  move.Action := ACTION_MAGIC_MISSILE;
end;

end.
```

3.3.7 Ruby example

In Ruby current wizard's name is changed from «`self`» to «`me`».

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

class MyStrategy
  # @param [Wizard] me
  # @param [World] world
  # @param [Game] game
  # @param [Move] move
  def move(me, world, game, move)
    move.speed = game.wizard_forward_speed
    move.speed = game.wizard_strafe_speed
    move.turn = game.wizard_max_turn_angle
    move.action = ActionType::MAGIC_MISSILE
  end
end
```

Глава 4

Package model

Package Contents *Page*

4.1 Classes

4.1.1 CLASS **ActionType**

Available wizard actions.

A wizard can not perform any new action, if he is not yet recovered from his previous action (`wizard.remainingActionCooldownTicks` is greater than `0`).

A wizard can not perform the specific new action, if it is not yet recovered from its previous usage (`remainingCooldownTicksByAction[actionType.ordinal()]` is greater than `0`).

DECLARATION

public final class ActionType

extends Enum

FIELDS

- public static final ActionType NONE
  - Do nothing.
- public static final ActionType STAFF
  - Perform a melee attack with a staff.

    This attack damages all living units in a sector of `-game.staffSector / 2.0` to `game.staffSector / 2.0`. The distance between wizard and target centers should not exceed `game.staffRange + livingUnit.radius`.
- public static final ActionType MAGIC_MISSILE
  - Cast a magic missile.

    Magic missile is a basic spell of any wizard. Inflicts some damage upon a direct hit.

    The center of a newly created magic missile is the same as the center of a caster wizard. The angle of a projectile is equal to `wizard.angle + move.castAngle`, and its speed is `game.magicMissileSpeed`. All collisions between a projectile and its caster are ignored by the game engine.

    Requires `game.magicMissileManacost` manapoints.
- public static final ActionType FROST_BOLT
  - Cast a frost bolt.

    A frost bolt inflicts some damage upon a direct hit and freezes a target.

The center of a newly created frost bolt is the same as the center of a caster wizard. The angle of a projectile is equal to `wizard.angle + move.castAngle`, and its speed is `game.frostBoltSpeed`. All collisions between a projectile and its caster are ignored by the game engine.

Requires the `FROST_BOLT` skill and `game.frostBoltManacost` manapoints.

- public static final ActionType FIREBALL
  - Cast a fireball.

    A fireball explodes when reaching maximal cast range or upon a collision with living unit. Damages and burns all living units nearby.

    The center of a newly created fireball is the same as the center of a caster wizard. The angle of a projectile is equal to `wizard.angle + move.castAngle`, and its speed is `game.fireballSpeed`. All collisions between a projectile and its caster are ignored by the game engine.

    Requires the `FIREBALL` skill and `game.fireballManacost` manapoints.
- public static final ActionType HASTE
  - Cast a haste spell, that temporarily speedups the friendly wizard with ID equal to `move.statusTargetId` or the caster himself if the game engine can not find such wizard.

    Requires the `HASTE` skill and `game.hasteManacost` manapoints.
- public static final ActionType SHIELD
  - Cast a shield spell, that temporarily protects the friendly wizard with ID equal to `move.statusTargetId` or the caster himself if the game engine can not find such wizard.

    Requires the `SHIELD` skill and `game.shieldManacost` manapoints.

4.1.2 CLASS **Bonus**

This class describes a bonus. Bonus is a static useful circular unit.

DECLARATION

public class Bonus **extends** CircularUnit

METHODS

- *getType*
  `public BonusType` **`getType( )`**
  - **Returns** - the bonus type.

4.1.3 CLASS **BonusType**

Bonus type.

Besides the primary effect each taken bonus gives `game.bonusScoreAmount` score points to the player. The wizard gains the same amount as xp.

DECLARATION

public final class BonusType

extends Enum

FIELDS

- public static final BonusType EMPOWER
  - Dramatically increases the damage of ranged and melee attacks for some time.
- public static final BonusType HASTE
  - Grants the `HASTENED` status to the wizard.

    Duration of the status is longer than usually.
- public static final BonusType SHIELD
  - Grants the `SHIELDED` status to the wizard.

    Duration of the status is longer than usually.

4.1.4 CLASS **Building**

This class describes a building. Faction building automatically attack a random enemy in a certain range.

A building can not be (`FROZEN`).

DECLARATION

public class Building

extends LivingUnit

METHODS

- *getAttackRange*
  `public double` **getAttackRange**`( )`
  - **Returns** - the maximal range (between units' centers), at which this building can attack other units.
- *getCooldownTicks*
  `public int` **getCooldownTicks**`( )`
  - **Returns** - the delay between attacks.
- *getDamage*
  `public int` **getDamage**`( )`
  - **Returns** - the damage of one attack.
- *getRemainingActionCooldownTicks*
  `public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - the amount of ticks remaining before the next attack.
- *getType*
  `public BuildingType` **getType**`( )`
  - **Returns** - the building type.
- *getVisionRange*
  `public double` **getVisionRange**`( )`
  - **Returns** - the maximal range (between units' centers), at which this building can detect other units.

4.1.5 CLASS **BuildingType**

Building type.

DECLARATION

public final class BuildingType

extends Enum

FIELDS

- public static final BuildingType GUARDIAN_TOWER
  - Guardian tower.
- public static final BuildingType FACTION_BASE
  - Faction base.

4.1.6 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.

4.1.7 CLASS **Faction**

Unit or player faction.

DECLARATION

public final class Faction

extends Enum

FIELDS

- public static final Faction ACADEMY
  - Wizards, minions and buildings of Academy.
- public static final Faction RENEGADES
  - Wizards, minions and buildings of Renegades.
- public static final Faction NEUTRAL
  - Neutral units. Do not attack first, but will strike back when damaged.
- public static final Faction OTHER
  - All other units in the game world.

4.1.8 CLASS **Game**

An instance of this class contains all game constants.

DECLARATION

public class Game

extends Object

METHODS

- *getAuraSkillRange*
  `public double` **getAuraSkillRange**`( )`
  - **Returns** - the range of an aura skill.

- *getBonusAppearanceIntervalTicks*
  `public int` **getBonusAppearanceIntervalTicks**`( )`
  - **Returns** - the interval at which appears a bonus.

    Every 2500 ticks up to two bonuses may appear on the map. The bonuses are created at the two following points: (`mapSize * 0.3`, `mapSize * 0.3`) and (`mapSize * 0.7`, `mapSize * 0.7`). If any part of the new bonus area is already occupied by a wizard or by an existing bonus, then the creation of the bonus will be postponed till the end of the next interval.

- *getBonusRadius*
  `public double` **getBonusRadius**`( )`
  - **Returns** - the radius of a bonus.

- *getBonusScoreAmount*
  `public int` **getBonusScoreAmount**`( )`
  - **Returns** - the amount of score and experience points for taking a bonus.

- *getBuildingDamageScoreFactor*
  `public double` **getBuildingDamageScoreFactor**`( )`
  - **Returns** - the factor of the experience points gained by the wizard for the damage dealt to the opposite faction buildings.

- *getBuildingEliminationScoreFactor*
  `public double` **getBuildingEliminationScoreFactor**`( )`
  - **Returns** - the factor of the experience points gained by the wizard for destroying the opposite faction building.

    Applies to the maximal amount of building's hitpoints.

- *getBurningDurationTicks*
  `public int` **getBurningDurationTicks**`( )`

  — **Returns** - the duration of the `BURNING` status.

- *getBurningSummaryDamage*
  `public int` **getBurningSummaryDamage**`( )`

  — **Returns** - the total damage of the `BURNING` status.

- *getDartDirectDamage*
  `public int` **getDartDirectDamage**`( )`

  — **Returns** - the dart damage.

- *getDartRadius*
  `public double` **getDartRadius**`( )`

  — **Returns** - the radius of a dart.

- *getDartSpeed*
  `public double` **getDartSpeed**`( )`

  — **Returns** - the dart speed.

- *getEmpoweredDamageFactor*
  `public double` **getEmpoweredDamageFactor**`( )`

  — **Returns** - the damage multiplier of empowered living unit. DOT (damage over time) is excluded.

- *getEmpoweredDurationTicks*
  `public int` **getEmpoweredDurationTicks**`( )`

  — **Returns** - the duration of the `EMPOWERED` status.

- *getFactionBaseAttackRange*
  `public double` **getFactionBaseAttackRange**`( )`

  — **Returns** - the maximal range (between units' centers), at which a faction base can attack other units.

- *getFactionBaseCooldownTicks*
  `public int` **getFactionBaseCooldownTicks**`( )`

  — **Returns** - the minimal possible interval between any two attacks of a faction base.

- *getFactionBaseDamage*
  `public int` **getFactionBaseDamage**`( )`

  — **Returns** - the damage of one attack of a faction base.

- *getFactionBaseLife*
  `public double` **getFactionBaseLife**`( )`

  — **Returns** - the maximal amount of faction base's hitpoints.

- *getFactionBaseRadius*
  `public double` **getFactionBaseRadius**`( )`

  — **Returns** - the radius of a faction base.

- *getFactionBaseVisionRange*
  `public double` **getFactionBaseVisionRange**`( )`

  — **Returns** - the maximal range (between units' centers), at which a faction base can detect other units.

- *getFactionMinionAppearanceIntervalTicks*
`public int` **getFactionMinionAppearanceIntervalTicks**`( )`
  - **Returns** - the interval at which appear the minions of the two opposing factions (`ACADEMY` and `RENEGADES`).

    The minions of each of these factions appear in three groups near their base (one group per lane). Each group consists of three orcs and one fetish. The minions immediately begin to advance on their lane toward the opposite faction base, attacking all enemies in their path.

- *getFetishBlowdartActionCooldownTicks*
`public int` **getFetishBlowdartActionCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two attacks of a fetish.

- *getFetishBlowdartAttackRange*
`public double` **getFetishBlowdartAttackRange**`( )`
  - **Returns** - the maximal dart fly range.

- *getFetishBlowdartAttackSector*
`public double` **getFetishBlowdartAttackSector**`( )`
  - **Returns** - the dart throw sector.

    The relative angle of a dart is in range of `-fetishBlowdartAttackSector / 2.0` to `fetishBlowdartAttackSector / 2.0`.

- *getFireballCooldownTicks*
`public int` **getFireballCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two “Fireball” spell casts.

- *getFireballExplosionMaxDamage*
`public int` **getFireballExplosionMaxDamage**`( )`
  - **Returns** - the damage of the fireball at the epicenter of the explosion.

    A living unit takes the `fireballExplosionMaxDamage` if the distance from the center of the explosion to the nearest point of this unit does not exceed the `fireballExplosionMaxDamageRange`. As you increase the distance to the `fireballExplosionMinDamageRange`, the damage of the fireball decreases in a linear manner and reaches `fireballExplosionMinDamage`. If the distance from the center of the explosion to the nearest point of the living unit exceeds `fireballExplosionMinDamageRange`, this unit takes no damage.

    If a living unit takes any damage from the fireball explosion, it receives a `BURNING` status.

- *getFireballExplosionMaxDamageRange*
`public double` **getFireballExplosionMaxDamageRange**`( )`
  - **Returns** - the radius of the area in which living units are taking maximal damage from the fireball projectile explosion.
  - **See Also**
    * `Game.getFireballExplosionMaxDamage()`

- *getFireballExplosionMinDamage*
`public int` **getFireballExplosionMinDamage**`( )`
  - **Returns** - the damage of the fireball on the periphery of the explosion.
  - **See Also**
    * `Game.getFireballExplosionMaxDamage()`

- *getFireballExplosionMinDamageRange*
`public double` **getFireballExplosionMinDamageRange**`( )`
  - **Returns** - the radius of the area in which living units are taking any damage from the fireball projectile explosion.
  - **See Also**
    - `Game.getFireballExplosionMaxDamage()`

- *getFireballManacost*
`public int` **getFireballManacost**`( )`
  - **Returns** - the "Fireball" spell manacost.

- *getFireballRadius*
`public double` **getFireballRadius**`( )`
  - **Returns** - the radius of a fireball projectile.

- *getFireballSpeed*
`public double` **getFireballSpeed**`( )`
  - **Returns** - the fireball projectile speed.

- *getFriendlyFireDamageFactor*
`public double` **getFriendlyFireDamageFactor**`( )`
  - **Returns** - Returns the damage part dealt by the wizards one faction to each other as a result of friendly fire.

    The value depends on the game mode, but is always in range of `0.0` to `1.0`.

    Regardless of the game mode, wizards can not damage friendly minions and buildings.

- *getFrostBoltCooldownTicks*
`public int` **getFrostBoltCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two "Frost bolt" spell casts.

- *getFrostBoltDirectDamage*
`public int` **getFrostBoltDirectDamage**`( )`
  - **Returns** - the frost bolt projectile damage.

- *getFrostBoltManacost*
`public int` **getFrostBoltManacost**`( )`
  - **Returns** - the "Frost bolt" spell manacost.

- *getFrostBoltRadius*
`public double` **getFrostBoltRadius**`( )`
  - **Returns** - the radius of a frost bolt projectile.

- *getFrostBoltSpeed*
`public double` **getFrostBoltSpeed**`( )`
  - **Returns** - the frost bolt projectile speed.

- *getFrozenDurationTicks*
`public int` **getFrozenDurationTicks**`( )`
  - **Returns** - the duration of the `FROZEN` status.

- *getGuardianTowerAttackRange*
  `public double` **getGuardianTowerAttackRange**`( )`
  - **Returns** - the maximal range (between units' centers), at which a guardian tower can attack other units.

- *getGuardianTowerCooldownTicks*
  `public int` **getGuardianTowerCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two attacks of a guardian tower.

- *getGuardianTowerDamage*
  `public int` **getGuardianTowerDamage**`( )`
  - **Returns** - the damage of one attack of a guardian tower.

- *getGuardianTowerLife*
  `public double` **getGuardianTowerLife**`( )`
  - **Returns** - the maximal amount of guardian tower's hitpoints.

- *getGuardianTowerRadius*
  `public double` **getGuardianTowerRadius**`( )`
  - **Returns** - the radius of a guardian tower.

- *getGuardianTowerVisionRange*
  `public double` **getGuardianTowerVisionRange**`( )`
  - **Returns** - the maximal range (between units' centers), at which a guardian tower can detect other units.

- *getHasteCooldownTicks*
  `public int` **getHasteCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two "Haste" spell casts.

- *getHasteManacost*
  `public int` **getHasteManacost**`( )`
  - **Returns** - the "Haste" spell manacost.

- *getHastenedBonusDurationFactor*
  `public double` **getHastenedBonusDurationFactor**`( )`
  - **Returns** - the `HASTENED` status duration multiplier (in case of taking a bonus).

- *getHastenedDurationTicks*
  `public int` **getHastenedDurationTicks**`( )`
  - **Returns** - the duration of the `HASTENED` status.

- *getHastenedMovementBonusFactor*
  `public double` **getHastenedMovementBonusFactor**`( )`
  - **Returns** - the relative move speed boost of a hastened unit.

    The maximal possible wizard speed is `1.0 + 4.0 * movementBonusFactorPerSkillLevel + hastenedMovementBonusFactor` of the base speed.

- *getHastenedRotationBonusFactor*
  `public double` **getHastenedRotationBonusFactor**`( )`

- **Returns** - the relative turn speed boost of a hastened unit.

- *getLevelUpXpValues*
`public int[] getLevelUpXpValues( )`

  - **Returns** - the non-negative integers.

    The numbers of items is equal to the number of levels a wizard can gain. An item `N` mean a number of experience points a wizard of level `N` should get to reach the next level. Thus, the amount of experience required for the zero level wizard to get to the level `N`, is the sum of the first `N` elements.

- *getMagicalDamageAbsorptionPerSkillLevel*
`public int getMagicalDamageAbsorptionPerSkillLevel( )`

  - **Returns** - the absolute decrease of the incoming magical damage for each learned skill, which is one of the prerequisites of the `SHIELD` skill. DOT (damage over time) is excluded.

- *getMagicalDamageBonusPerSkillLevel*
`public int getMagicalDamageBonusPerSkillLevel( )`

  - **Returns** - the absolute increase of the wizard spell damage for each learned skill, which is one of the prerequisites of the `FROST_BOLT` skill. DOT (damage over time) is excluded.

- *getMagicMissileCooldownTicks*
`public int getMagicMissileCooldownTicks( )`

  - **Returns** - the minimal possible interval between any two "Magic missile" spell casts.

- *getMagicMissileDirectDamage*
`public int getMagicMissileDirectDamage( )`

  - **Returns** - the magic missile projectile damage.

- *getMagicMissileManacost*
`public int getMagicMissileManacost( )`

  - **Returns** - the "Magic missile" spell manacost.

- *getMagicMissileRadius*
`public double getMagicMissileRadius( )`

  - **Returns** - the radius of a magic missile projectile.

- *getMagicMissileSpeed*
`public double getMagicMissileSpeed( )`

  - **Returns** - the magic missile projectile speed.

- *getMapSize*
`public double getMapSize( )`

  - **Returns** - the size (both width and height) of the map.

- *getMinionDamageScoreFactor*
`public double getMinionDamageScoreFactor( )`

  - **Returns** - the factor of the experience points gained by the wizard for the damage dealt to the other faction minions.

- *getMinionEliminationScoreFactor*
`public double getMinionEliminationScoreFactor( )`

- **Returns** - the factor of the experience points gained by the wizard for killing the other faction minion.

  Applies to the maximal amount of minion's hitpoints.

- *getMinionLife*
`public int` **getMinionLife**`( )`

  - **Returns** - the maximal amount of minion's hitpoints.

- *getMinionMaxTurnAngle*
`public double` **getMinionMaxTurnAngle**`( )`

  - **Returns** - the maximal turn speed of a minion.

- *getMinionRadius*
`public double` **getMinionRadius**`( )`

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

- *getMinionSpeed*
`public double` **getMinionSpeed**`( )`

  - **Returns** - the forward speed of a minion.

    A minion can not strafe or move backward.

- *getMinionVisionRange*
`public double` **getMinionVisionRange**`( )`

  - **Returns** - the maximal range (between units' centers), at which a minion can detect other units.

- *getMovementBonusFactorPerSkillLevel*
`public double` **getMovementBonusFactorPerSkillLevel**`( )`

  - **Returns** - the relative increase of the move speed for each learned skill, which is one of the prerequisites of the `HASTE` skill.

    The maximal possible wizard speed is `1.0 + 4.0 * movementBonusFactorPerSkillLevel + hastenedMovementBonusFactor` of the base speed.

- *getOrcWoodcutterActionCooldownTicks*
`public int` **getOrcWoodcutterActionCooldownTicks**`( )`

  - **Returns** - the minimal possible interval between any two attacks of an orc.

- *getOrcWoodcutterAttackRange*
`public double` **getOrcWoodcutterAttackRange**`( )`

  - **Returns** - the orc's axe range.

    An axe attack damages all living units if the distance between orc's and target's centers is not greater than `orcWoodcutterAttackRange + livingUnit.radius`.

- *getOrcWoodcutterAttackSector*
`public double` **getOrcWoodcutterAttackSector**`( )`

  - **Returns** - the orc's axe sector.

    An axe attack damages all living units in a sector of `-orcWoodcutterAttackSector / 2.0` to `orcWoodcutterAttackSector / 2.0`.

- *getOrcWoodcutterDamage*
  `public int` **getOrcWoodcutterDamage**`( )`
  - **Returns** - the damage of one attack of an orc.

- *getRandomSeed*
  `public long` **getRandomSeed**`( )`
  - **Returns** - the number that your strategy may use to initialize a generator of random numbers.

- *getRangeBonusPerSkillLevel*
  `public double` **getRangeBonusPerSkillLevel**`( )`
  - **Returns** - the absolute increase of the wizard cast range for each learned skill, which is one of the prerequisites of the `ADVANCED_MAGIC_MISSILE` skill.

- *getRawMessageMaxLength*
  `public int` **getRawMessageMaxLength**`( )`
  - **Returns** - the maximal possible length of a raw message.

    If a message has higher length, then it will be completely ignored.

- *getRawMessageTransmissionSpeed*
  `public double` **getRawMessageTransmissionSpeed**`( )`
  - **Returns** - the raw message transmission speed.

    If the raw message is empty, the addressee will receive it in the next game tick. In other case, the time of receipt of the message will be delayed for `ceil(message.rawMessage.length / rawMessageTransmissionSpeed)` game ticks.

- *getScoreGainRange*
  `public double` **getScoreGainRange**`( )`
  - **Returns** - the maximal range, at which a wizard gains experience points in case, if a friendly unit kills an other faction unit.

    The experience is evenly distributed between all the wizards not farther than `scoreGainRange` from a killed unit, and the killer unit if he is also a wizard.

    If the damage is not fatal, this parameter is not used. If the attacker is a wizard, than the experience completely goes to this wizard. If the attacker is a minion or a building, nobody gains an experience.

    The range is considered as the range between units' centers.

- *getShieldCooldownTicks*
  `public int` **getShieldCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two "Shield" spell casts.

- *getShieldedBonusDurationFactor*
  `public double` **getShieldedBonusDurationFactor**`( )`
  - **Returns** - the `SHIELDED` status duration multiplier (in case of taking a bonus).

- *getShieldedDirectDamageAbsorptionFactor*
  `public double` **getShieldedDirectDamageAbsorptionFactor**`( )`
  - **Returns** - the damage part absorbed by shield. DOT (damage over time) is excluded.

- *getShieldedDurationTicks*
  `public int` **getShieldedDurationTicks**`( )`
  - **Returns** - the `SHIELDED` duration.

- *getShieldManacost*
  `public int` **getShieldManacost**`( )`
  - **Returns** - the “Shield” spell manacost.

- *getStaffCooldownTicks*
  `public int` **getStaffCooldownTicks**`( )`
  - **Returns** - the minimal possible interval between any two staff attacks.

- *getStaffDamage*
  `public int` **getStaffDamage**`( )`
  - **Returns** - the base staff damage.

    The effective damage may be higher depending on skills of the wizard and auras of nearby friendly wizards.

- *getStaffDamageBonusPerSkillLevel*
  `public int` **getStaffDamageBonusPerSkillLevel**`( )`
  - **Returns** - the absolute increase of the wizard staff damage for each learned skill, which is one of the prerequisites of the `FIREBALL` skill.

- *getStaffRange*
  `public double` **getStaffRange**`( )`
  - **Returns** - the wizard’s staff range.

    A staff attack damages all living units if the distance between wizard’s and target’s centers is not greater than `staffRange + livingUnit.radius`.

- *getStaffSector*
  `public double` **getStaffSector**`( )`
  - **Returns** - the wizard’s staff sector.

    A staff attack damages all living units in a sector of `-staffSector / 2.0` to `staffSector / 2.0`. This also applies to the status spells and to the relative projectile angle.

- *getTeamWorkingScoreFactor*
  `public double` **getTeamWorkingScoreFactor**`( )`
  - **Returns** - the experience multiplier applied in case, if the enemy unit died near two or more friendly wizards.

    After applying this multiplier, the amount of the experience is rounded down.

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

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

- **Returns** - the amount of experience points received by each player of the faction in case of victory.

- *getWizardActionCooldownTicks*
`public int` **getWizardActionCooldownTicks**`( )`

  - **Returns** - the minimal possible interval between any two actions of a wizard.

- *getWizardBackwardSpeed*
`public double` **getWizardBackwardSpeed**`( )`

  - **Returns** - the base limit of wizard's backward speed.

    The effective backward speed may be higher depending on skills of the wizard and auras of nearby friendly wizards. The `HASTENED` status can also greatly speed up a wizard.

- *getWizardBaseLife*
`public int` **getWizardBaseLife**`( )`

  - **Returns** - the maximal amount of wizard's hitpoints at initial level.

- *getWizardBaseLifeRegeneration*
`public double` **getWizardBaseLifeRegeneration**`( )`

  - **Returns** - the regeneration speed of wizard's hitpoints at initial level.

- *getWizardBaseMana*
`public int` **getWizardBaseMana**`( )`

  - **Returns** - the maximal amount of wizard's manapoints at initial level.

- *getWizardBaseManaRegeneration*
`public double` **getWizardBaseManaRegeneration**`( )`

  - **Returns** - the regeneration speed of wizard's manapoints at initial level.

- *getWizardCastRange*
`public double` **getWizardCastRange**`( )`

  - **Returns** - the base cast range of a wizard.

    The effective cast range (`wizard.castRange`) may be higher depending on skills of the wizard and auras of nearby friendly wizards.

- *getWizardDamageScoreFactor*
`public double` **getWizardDamageScoreFactor**`( )`

  - **Returns** - the factor of the experience points gained by the wizard for the damage dealt to the opposite faction wizards.

- *getWizardEliminationScoreFactor*
`public double` **getWizardEliminationScoreFactor**`( )`

  - **Returns** - the factor of the experience points gained by the wizard for killing the opposite faction wizard.

    Applies to the maximal amount of wizard's hitpoints.

- *getWizardForwardSpeed*
`public double` **getWizardForwardSpeed**`( )`

  - **Returns** - the base limit of wizard's forward speed.

    The effective forward speed may be higher depending on skills of the wizard and auras of nearby friendly wizards. The `HASTENED` status can also greatly speed up a wizard.

- *getWizardLifeGrowthPerLevel*
`public int` **getWizardLifeGrowthPerLevel**`( )`

  — **Returns** - the growth of wizard's hitpoints per level.

- *getWizardLifeRegenerationGrowthPerLevel*
`public double` **getWizardLifeRegenerationGrowthPerLevel**`( )`

  — **Returns** - the growth of the regeneration speed of wizard's hitpoints.

- *getWizardManaGrowthPerLevel*
`public int` **getWizardManaGrowthPerLevel**`( )`

  — **Returns** - the growth of wizard's manapoints per level.

- *getWizardManaRegenerationGrowthPerLevel*
`public double` **getWizardManaRegenerationGrowthPerLevel**`( )`

  — **Returns** - the growth of the regeneration speed of wizard's manapoints.

- *getWizardMaxResurrectionDelayTicks*
`public int` **getWizardMaxResurrectionDelayTicks**`( )`

  — **Returns** - the maximal possible delay of a wizard's revival.

- *getWizardMaxTurnAngle*
`public double` **getWizardMaxTurnAngle**`( )`

  — **Returns** - the base limit of wizard's turn speed.

  The `HASTENED` status increases this limit by `1.0 + hastenedRotationBonusFactor` times.

- *getWizardMinResurrectionDelayTicks*
`public int` **getWizardMinResurrectionDelayTicks**`( )`

  — **Returns** - the minimal possible delay of a wizard's revival.

- *getWizardRadius*
`public double` **getWizardRadius**`( )`

  — **Returns** - the radius of a wizard.

- *getWizardStrafeSpeed*
`public double` **getWizardStrafeSpeed**`( )`

  — **Returns** - the base limit of wizard's strafe speed.

  The effective strafe speed may be higher depending on skills of the wizard and auras of nearby friendly wizards. The `HASTENED` status can also greatly speed up a wizard.

- *getWizardVisionRange*
`public double` **getWizardVisionRange**`( )`

  — **Returns** - the maximal range (between units' centers), at which a wizard can detect other units.

- *isRawMessagesEnabled*
`public boolean` **isRawMessagesEnabled**`( )`

  — **Returns** - `true` if and only if the master wizards in this game can send raw messages.

- *isSkillsEnabled*
`public boolean` **isSkillsEnabled**`( )`

  — **Returns** - `true` if and only if the wizards in this game can gain new levels and learn skills.

4.1.9 CLASS **LaneType**

Lane type.

DECLARATION

public final class LaneType

extends Enum

FIELDS

- public static final LaneType TOP
  - Top lane. It goes through the lower left, the upper left and the upper right corners of the map.
- public static final LaneType MIDDLE
  - Middle lane. Directly connects the lower left and the upper right corners of the map.
- public static final LaneType BOTTOM
  - Bottom lane. It goes through the lower left, the lower right and the upper right corners of the map.

4.1.10 CLASS **LivingUnit**

This base class is inherited from a circular unit and describes any living unit in the game world.

DECLARATION

public abstract class LivingUnit

extends CircularUnit

METHODS

- *getLife*
  `public int getLife( )`
  - **Returns** - the current amount of hitpoints.

- *getMaxLife*
  `public int` **getMaxLife**`( )`
  - **Returns** - the maximal amount of hitpoints.

- *getStatuses*
  `public Status[]` **getStatuses**`( )`
  - **Returns** - the magical statuses affecting this living unit.

4.1.11 CLASS **Message**

This class describes a message, that master wizard can send to other wizards of his faction.

The message is sent personally to each wizard. Other wizards are unable to intercept him.

The recipient receives the message in the next game tick or later, depending on the size of the message.

The wizard is free to ignore as any part of the message and the entire message, however this can lead to the defeat of wizard's faction.

DECLARATION

public class Message **extends** Object

METHODS

- *getLane*
  `public LaneType` **getLane**`( )`
  - **Returns** - the order to control the specified lane.

- *getRawMessage*
  `public byte[]` **getRawMessage**`( )`
  - **Returns** - the text message in a forgotten ancient language.

    The maximal message length is `game.rawMessageMaxLength`. The speed of sending a message depends on its length. If the text part of the message is empty, the addressee will receive it in the next game tick. In other case, the time of receipt of the message will be delayed for `ceil(rawMessage.length / game.rawMessageTransmissionSpeed)` game ticks.

    The field value may not be available in all game modes.

- *getSkillToLearn*
  `public SkillType` **getSkillToLearn**`( )`

– **Returns** - the order to learn the specified skill.

This skill may require to learn other skills or be unavailable for learning at the moment due to the low level. The wizard should remember the order and move towards its achievement. The later the order, the higher the priority.

The field value may not be available in all game modes.

4.1.12 CLASS **Minion**

This class is inherited from a living unit and describes a minion.

DECLARATION

public class Minion **extends** LivingUnit

METHODS

- *getCooldownTicks*
  `public int` **getCooldownTicks**`( )`
  – **Returns** - the delay between attacks.
- *getDamage*
  `public int` **getDamage**`( )`
  – **Returns** - the damage of one attack.
- *getRemainingActionCooldownTicks*
  `public int` **getRemainingActionCooldownTicks**`( )`
  – **Returns** - the amount of ticks remaining before the next attack.
- *getType*
  `public MinionType` **getType**`( )`
  – **Returns** - the minion type.
- *getVisionRange*
  `public double` **getVisionRange**`( )`
  – **Returns** - the maximal range (between units' centers), at which this minion can detect other units.

4.1.13 CLASS **MinionType**

Minion type.

DECLARATION

public final class MinionType

extends Enum

FIELDS

- public static final MinionType ORC_WOODCUTTER
  - A melee fighter. No so strong as orc warrior, but still dangerous.
- public static final MinionType FETISH_BLOWDART
  - A magical creature with sharp darts.

4.1.14 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 wizard action.
- *getCastAngle*
  `public double` **getCastAngle**`( )`
  - **Returns** - the current cast angle.
- *getMaxCastDistance*
  `public double` **getMaxCastDistance**`( )`
  - **Returns** - the current maximal cast distance.
- *getMessages*
  `public Message[]` **getMessages**`( )`

- **Returns** - the current messages for friendly wizards.

- *getMinCastDistance*
`public double` **getMinCastDistance**`( )`

  - **Returns** - the current minimal cast distance.

- *getSkillToLearn*
`public SkillType` **getSkillToLearn**`( )`

  - **Returns** - the currently selected skill to learn.

- *getSpeed*
`public double` **getSpeed**`( )`

  - **Returns** - the current move speed.

- *getStatusTargetId*
`public long` **getStatusTargetId**`( )`

  - **Returns** - the current ID of the status spell target.

- *getStrafeSpeed*
`public double` **getStrafeSpeed**`( )`

  - **Returns** - the current strafe speed.

- *getTurn*
`public double` **getTurn**`( )`

  - **Returns** - the current turn angle.

- *setAction*
`public void` **setAction**`( ActionType` **action** `)`

  - **Usage**
    * Sets the action for one tick.

      The specified action can be ignored by the game engine, if the controlling wizard has insufficient manapoints or this action is on cooldown.

- *setCastAngle*
`public void` **setCastAngle**`( double` **castAngle** `)`

  - **Usage**
    * Sets the cast angle for one tick.

      The cast angle is in radians and is relative to the current angle of the wizard. The cast angle is in range of `-game.staffSector / 2.0` to `game.staffSector / 2.0`.

      If a specified value is out of the range, than it become equal to the nearest value of the range. The positive values mean turning clockwise.

      If the specified action is not a projectile spell, than the game engine will simply ignore this parameter.

- *setMaxCastDistance*
`public void` **setMaxCastDistance**`( double` **maxCastDistance** `)`

  - **Usage**

* Sets the maximal cast distance for one tick.

  If the distance from the center of the projectile to the point of its occurrence is greater than the value of this parameter, the projectile will be removed from the game world. In this case, the `FIREBALL` projectile detonates.

  The default value of this parameter is higher than the maximal flying range of any projectile in the game.

  If the specified action is not a projectile spell, than the game engine will simply ignore this parameter.

- *setMessages*
  `public void` **setMessages**`( Message[]` **messages** `)`
  - **Usage**
    * Sets the messages for the wizards of the same faction.

      Available only to the master wizard. If not empty, the number of messages must be strictly equal to the number of wizards of the friendly faction (dead or alive) except the master wizard.

      Messages are addressed in ascending order of wizard IDs. Some messages can be empty (`null`), if supported by the programming language used by strategy. In other case all items should be the correct messages.

      The game engine may ignore the message to a specific wizard, if there is another pending message to the same wizard. If the addressed wizard is dead, then the message will be removed from the game world and the wizard will never get it.

      Not all game modes support the messages.

- *setMinCastDistance*
  `public void` **setMinCastDistance**`( double` **minCastDistance** `)`
  - **Usage**
    * Sets the minimal cast distance for one tick.

      If the distance from the center of the projectile to the point of its occurrence is less than the value of this parameter, the battle properties of the projectile are ignored. The projectile passes freely through all other game objects, except trees.

      Default value is `0.0`. All collisions between a projectile and its caster are ignored by the game engine.

      If the specified action is not a projectile spell, than the game engine will simply ignore this parameter.

- *setSkillToLearn*
  `public void` **setSkillToLearn**`( SkillType` **skillToLearn** `)`
  - **Usage**
    * Selects the skill to learn before the start of the next tick.

      The setting will be ignored by the game engine if the current wizard level is less than or equal to the number of the already learned skills. Some skills may also require learning other skills.

      In some game modes a wizard can not learn skills.

- *setSpeed*
`public void` **setSpeed**`( double` **speed** `)`
  - **Usage**
    * Sets move speed for one tick.

      By default the speed is in range of `-game.wizardBackwardSpeed` to `game.wizardForwardSpeed`. These limits can be extended depending on skills of moving wizard and auras of nearby friendly wizards. The `HASTENED` status can also greatly speed up a wizard.

      If a specified value is out of the range, than it become equal to the nearest value of the range. The positive values mean moving forward.

      If the value `hypot(speed / maxSpeed, strafeSpeed / maxStrafeSpeed)` is greater than `1.0`, than both `speed` and `strafeSpeed` will be divided by this value.

- *setStatusTargetId*
`public void` **setStatusTargetId**`( long` **statusTargetId** `)`
  - **Usage**
    * Sets the ID of the target living unit to cast a status spell.

      According to the game rules, the valid targets are only the wizards of the same faction. If the wizard with the specified ID is not found, the status is applied directly to the wizard performing the action. The relative angle to the target should be in range of `-game.staffSector / 2.0` to `game.staffSector / 2.0`. The distance to the target is limited by `wizard.castRange`.

      The default value of this parameter is `-1` (wrong ID).

      If the specified action is not a status spell, than the game engine will simply ignore this parameter.

- *setStrafeSpeed*
`public void` **setStrafeSpeed**`( double` **strafeSpeed** `)`
  - **Usage**
    * Sets the strafe speed for one tick.

      By default the strafe speed is in range of `-game.wizardStrafeSpeed` to `game.wizardStrafeSpeed`. These limits can be extended depending on skills of moving wizard and auras of nearby friendly wizards. The `HASTENED` status can also greatly speed up a wizard.

      If a specified value is out of the range, than it become equal to the nearest value of the range. The positive values mean moving to the right.

      If the value `hypot(speed / maxSpeed, strafeSpeed / maxStrafeSpeed)` is greater than `1.0`, than both `speed` and `strafeSpeed` will be divided by this value.

- *setTurn*
`public void` **setTurn**`( double` **turn** `)`
  - **Usage**
    * Sets the turn angle for one tick.
      The turn angle is in radians and is relative to the current angle of the wizard. By default the turn angle is in range of `-game.wizardMaxTurnAngle` to `game.wizardMaxTurnAngle`. The `HASTENED` status increases bot limits by `1.0 + game.hastenedRotationBonusFactor` times.

      If a specified value is out of the range, than it become equal to the nearest value of the range. The positive values mean turning clockwise.

4.1.15 CLASS **Player**

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

DECLARATION

public class Player **extends** Object

METHODS

- *getFaction*
  `public Faction` **getFaction**`( )`
  - **Returns** - the faction of this player.
- *getId*
  `public long` **getId**`( )`
  - **Returns** - the unique player ID.
- *getName*
  `public String` **getName**`( )`
  - **Returns** - the name of the player.
- *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.

4.1.16 CLASS **Projectile**

This class is inherited from a circular unit and describes a projectile.

DECLARATION

public class Projectile

extends CircularUnit

METHODS

- *getOwnerPlayerId*
  `public long` **getOwnerPlayerId**`( )`
  - **Returns** - the ID of the player, which unit created this projectile, or `-1`.
- *getOwnerUnitId*
  `public long` **getOwnerUnitId**`( )`
  - **Returns** - the ID of the unit created this projectile.
- *getType*
  `public ProjectileType` **getType**`( )`
  - **Returns** - the type of the projectile.

4.1.17 CLASS **ProjectileType**

Projectile type.

DECLARATION

public final class ProjectileType

extends Enum

FIELDS

- public static final ProjectileType MAGIC_MISSILE
  - A small piece of pure energy, that inflicts damage to a living unit upon a direct hit.
- public static final ProjectileType FROST_BOLT
  - Inflicts damage upon a direct hit and freezes a target for `game.frozenDurationTicks`. A frozen unit can not move or perform any actions.
- public static final ProjectileType FIREBALL

- Explodes when reaching maximal cast range or upon a collision with living unit. Damages and burns any living unit, if a distance to the center of this unit is not greater than `game.fireballExplosionMinDamageRange + livingUnit.radius`. The greater the distance, the less the instant damage.

- public static final ProjectileType DART
  - Sharp thing flying at high speed. Inflicts damage upon a direct hit.

4.1.18 CLASS **SkillType**

Skill type. In some game modes a wizard can not learn skills (see `game.skillsEnabled`).

There is three skill groups: active, passive and auras.

- Active skills provide an ability to perform a new action, not available before.
- Passive skills are constantly improving some characteristic of the wizard for a certain value.
- Auras are constantly improving some characteristic of the wizard and all friendly wizards in the `game.auraSkillRange`.

DECLARATION

public final class SkillType

extends Enum

FIELDS

- public static final SkillType RANGE_BONUS_PASSIVE_1
  - Passive skill. Increases cast range by `game.rangeBonusPerSkillLevel`.
- public static final SkillType RANGE_BONUS_AURA_1
  - Aura. Increases cast range by `game.rangeBonusPerSkillLevel`.

    Requires `RANGE_BONUS_PASSIVE_1`.
- public static final SkillType RANGE_BONUS_PASSIVE_2
  - Passive skill. Increases cast range by `2.0 * game.rangeBonusPerSkillLevel`.

    Requires `RANGE_BONUS_AURA_1`.
- public static final SkillType RANGE_BONUS_AURA_2
  - Aura. Increases cast range by `2.0 * game.rangeBonusPerSkillLevel`.

    Requires `RANGE_BONUS_PASSIVE_2`.
- public static final SkillType ADVANCED_MAGIC_MISSILE

  - Passive skill. Removes the `MAGIC_MISSILE` spell delay. The common action delay `game.wizardActionCooldownTicks` still applies.

    Requires `RANGE_BONUS_AURA_2`.

- public static final SkillType MAGICAL_DAMAGE_BONUS_PASSIVE_1
  - Passive skill. Increases instant magical damage by `game.magicalDamageBonusPerSkillLevel`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_AURA_1
  - Aura. Increases instant magical damage by `game.magicalDamageBonusPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_BONUS_PASSIVE_1`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_PASSIVE_2
  - Passive skill. Increases instant magical damage by `2.0 * game.magicalDamageBonusPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_BONUS_AURA_1`.
- public static final SkillType MAGICAL_DAMAGE_BONUS_AURA_2
  - Aura. Increases instant magical damage by `2.0 * game.magicalDamageBonusPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_BONUS_PASSIVE_2`.
- public static final SkillType FROST_BOLT
  - Active skill. A wizard can now use the `FROST_BOLT` spell.

    Requires `MAGICAL_DAMAGE_BONUS_AURA_2`.
- public static final SkillType STAFF_DAMAGE_BONUS_PASSIVE_1
  - Passive skill. Increases staff damage by `game.staffDamageBonusPerSkillLevel`.
- public static final SkillType STAFF_DAMAGE_BONUS_AURA_1
  - Aura. Increases staff damage by `game.staffDamageBonusPerSkillLevel`.

    Requires `STAFF_DAMAGE_BONUS_PASSIVE_1`.
- public static final SkillType STAFF_DAMAGE_BONUS_PASSIVE_2
  - Passive skill. Increases staff damage by `2.0 * game.staffDamageBonusPerSkillLevel`.

    Requires `STAFF_DAMAGE_BONUS_AURA_1`.
- public static final SkillType STAFF_DAMAGE_BONUS_AURA_2
  - Aura. Increases staff damage by `2.0 * game.staffDamageBonusPerSkillLevel`.

    Requires `STAFF_DAMAGE_BONUS_PASSIVE_2`.
- public static final SkillType FIREBALL
  - Active skill. A wizard can now use the `FIREBALL` spell.

    Requires `STAFF_DAMAGE_BONUS_AURA_2`.
- public static final SkillType MOVEMENT_BONUS_FACTOR_PASSIVE_1
  - Passive skill. Increases movement speed by `1.0 + game.movementBonusFactorPerSkillLevel` times.

    Summarily `MOVEMENT_BONUS_FACTOR_PASSIVE_2` and `MOVEMENT_BONUS_FACTOR_AURA_2` increase movement speed by `1.0 + 4.0 * game.movementBonusFactorPerSkillLevel` times.

- public static final SkillType MOVEMENT_BONUS_FACTOR_AURA_1
  - Aura. Increases movement speed by `1.0 + game.movementBonusFactorPerSkillLevel` times.

    Requires `MOVEMENT_BONUS_FACTOR_PASSIVE_1`.
- public static final SkillType MOVEMENT_BONUS_FACTOR_PASSIVE_2
  - Passive skill. Increases movement speed by `1.0 + 2.0 * game.movementBonusFactorPerSkillLevel` times.

    Requires `MOVEMENT_BONUS_FACTOR_AURA_1`.
- public static final SkillType MOVEMENT_BONUS_FACTOR_AURA_2
  - Aura. Increases movement speed by `1.0 + 2.0 * game.movementBonusFactorPerSkillLevel` times.

    Requires `MOVEMENT_BONUS_FACTOR_PASSIVE_2`.
- public static final SkillType HASTE
  - Active skill. A wizard can now use the `HASTE` spell.

    Requires `MOVEMENT_BONUS_FACTOR_AURA_2`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_PASSIVE_1
  - Passive skill. Decreases received magical damage by `game.magicalDamageAbsorptionPerSkillLevel`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_AURA_1
  - Aura. Decreases received magical damage by `game.magicalDamageAbsorptionPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_ABSORPTION_PASSIVE_1`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_PASSIVE_2
  - Passive skill. Decreases received magical damage by `2.0 * game.magicalDamageAbsorptionPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_ABSORPTION_AURA_1`.
- public static final SkillType MAGICAL_DAMAGE_ABSORPTION_AURA_2
  - Aura. Decreases received magical damage by `2.0 * game.magicalDamageAbsorptionPerSkillLevel`.

    Requires `MAGICAL_DAMAGE_ABSORPTION_PASSIVE_2`.
- public static final SkillType SHIELD
  - Active skill. A wizard can now use the `SHIELD` spell.

    Requires `MAGICAL_DAMAGE_ABSORPTION_AURA_2`.

4.1.19 CLASS **Status**

A magical status, affecting living unit.

DECLARATION

public class Status

extends Object

METHODS

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

- *getPlayerId*
  `public long` **getPlayerId**`( )`
  - **Returns** - the ID of the player, which unit casted this status, or `-1`.

- *getRemainingDurationTicks*
  `public int` **getRemainingDurationTicks**`( )`
  - **Returns** - the remaining status duration.

- *getType*
  `public StatusType` **getType**`( )`
  - **Returns** - the status type.

- *getWizardId*
  `public long` **getWizardId**`( )`
  - **Returns** - the ID of the wizard casted this status or `-1`.

4.1.20 CLASS StatusType

Status type.

DECLARATION

public final class StatusType

extends Enum

- public static final StatusType BURNING
  - A living unit receives some damage each time tick.
- public static final StatusType EMPOWERED
  - A living unit inflicts more damage than usually, excluding DOT (damage over time).
- public static final StatusType FROZEN
  - A living unit can not move or perform any actions.
- public static final StatusType HASTENED
  - A living unit has increased move and turn speed.
- public static final StatusType SHIELDED
  - A living unit receives less damage than usually, excluding DOT (damage over time).

4.1.21 CLASS **Tree**

This class is inherited from a living unit and describes a tree.

DECLARATION

public class Tree

extends LivingUnit

4.1.22 CLASS **Unit**

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

DECLARATION

public abstract class Unit

extends Object

- *getAngle*
  `public final double` **getAngle**`( )`
  - **Returns** - the turn angle in radians of this unit. Direction of the X-axis has zero angle. Positive angle corresponds to the rotation in a clockwise direction.

- *getAngleTo*
  `public double` **getAngleTo**`( double x, double y )`
  - **Parameters**
    - `x` - X-coordinate of the point to get the angle to.
    - `y` - Y-coordinate of the point to get the angle to.
  - **Returns** - the relative angle to the specified point. The angle is in range of `-PI` to `PI` both inclusive.

- *getAngleTo*
  `public double` **getAngleTo**`( Unit unit )`
  - **Parameters**
    - `unit` - the unit to get the angle to.
  - **Returns** - the relative angle to the center of the specified unit. The angle is in range of `-PI` to `PI` both inclusive.

- *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.

- *getFaction*
  `public Faction` **getFaction**`( )`
  - **Returns** - the faction of this unit.

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

- *getSpeedX*
  `public final double` **getSpeedX**`( )`
  - **Returns** - the X speed component or the last tick X-coordinate change, if this unit can instantly change its speed. The X-axis is directed from left to right.

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

- **Returns** - the Y speed component or the last tick Y-coordinate change, if this unit can instantly change its speed. The Y-axis is directed downward.

- *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.

4.1.23 CLASS **Wizard**

This class is inherited from a living unit and describes a wizard.

DECLARATION

public class Wizard **extends** LivingUnit

METHODS

- *getCastRange*
`public double` **`getCastRange( )`**
  - **Returns** - the maximal cast range..

- *getLevel*
`public int` **`getLevel( )`**
  - **Returns** - the current wizard level.

    Each wizard starts at level `0` and can level up up to `game.levelUpXpValues.length` times.

    In some game modes a wizard can not gain new levels.

- *getMana*
`public int` **`getMana( )`**
  - **Returns** - the current amount of manapoints.

- *getMaxMana*
`public int` **`getMaxMana( )`**
  - **Returns** - the maximal amount of manapoints.

- *getMessages*
`public Message[]` **`getMessages( )`**

- **Returns** - the messages addressed to this wizard.

  A strategy can only read messages of the controlling wizard.

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

- *getRemainingActionCooldownTicks*
`public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - the amount of ticks remaining before any new action.

    A wizard can perform the action `actionType` if and only if both `remainingActionCooldownTicks` and `remainingCooldownTicksByAction[actionType.ordinal()]` are equal to zero.

- *getRemainingCooldownTicksByAction*
`public int[]` **getRemainingCooldownTicksByAction**`( )`
  - **Returns** - the non-negative integer numbers. Each item is equal to the amount of ticks remaining before the next action with the corresponding index.

    For example, `remainingCooldownTicksByAction[0]` corresponds to `NONE` action and always equal to zero. `remainingCooldownTicksByAction[1]` corresponds to `STAFF` action and equal to the amount of ticks remaining before the next staff attack.

    A wizard can perform the action `actionType` if and only if both `remainingActionCooldownTicks` and `remainingCooldownTicksByAction[actionType.ordinal()]` are equal to zero.

- *getSkills*
`public SkillType[]` **getSkills**`( )`
  - **Returns** - the skills of this wizard.

- *getVisionRange*
`public double` **getVisionRange**`( )`
  - **Returns** - the maximal range (between units' centers), at which this wizard can detect other units.

- *getXp*
`public int` **getXp**`( )`
  - **Returns** - the current amount of experience points.

- *isMaster*
`public boolean` **isMaster**`( )`
  - **Returns** - `true` if and only if this wizard is master.

    There is exactly one master wizard per faction.

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

4.1.24 CLASS **World**

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

DECLARATION

public class World

extends Object

METHODS

- *getBonuses*
  `public Bonus[] getBonuses( )`
  - **Returns** - visible bonuses (in random order). After each tick the bonus objects are recreated.

- *getBuildings*
  `public Building[] getBuildings( )`
  - **Returns** - visible buildings (in random order). After each tick the building objects are recreated.

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

- *getMinions*
  `public Minion[] getMinions( )`
  - **Returns** - visible minions (in random order). After each tick the minion objects are recreated.

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

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

- *getProjectiles*
  `public Projectile[] getProjectiles( )`
  - **Returns** - visible projectiles (in random order). After each tick the projectile objects are recreated.

- *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.

- *getTrees*
  `public Tree[] getTrees( )`

- **Returns** - visible trees (in random order). After each tick the tree objects are recreated.

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

- *getWizards*
  `public Wizard[]` **getWizards**`( )`
  - **Returns** - visible wizards (in random order). After each tick the wizard objects are recreated.

Глава 5

Package <none>

Package Contents *Page*

Interfaces

This interface contains the methods that each wizard strategy should implement.

5.1 Interfaces

5.1.1 INTERFACE Strategy

This interface contains the methods that each wizard strategy should implement.

DECLARATION

public interface Strategy

METHODS

- *move*
  `public void` **move**`( Wizard` **self**`, World` **world**`, Game` **game**`, Move` **move** `)`
  - **Usage**
    - Main strategy method, controlling the wizard. The game engine calls this method once each time tick.
  - **Parameters**
    - `self` - the wizard controlling by this strategy.
    - `world` - the current world snapshot.
    - `game` - many game constants.
    - `move` - the object that encapsulates all strategy instructions to the `self` wizard.