МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | Автоматики и вычислительной техники |
| Кафедра | Автоматизированных систем управления |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  | Леонов Д.Г. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  | Папилина Т.М. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Системное программирование |
|  | |

|  |  |
| --- | --- |
| на тему | Турнир роботов. Разработка клиентского модуля |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛ: |  |
|  |  | Студент группы | **КИ-18-01** |
|  |  |  | (номер группы) |
|  |  | Ляскин Константин Алексеевич | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Москва, 20 | 21 |  |

МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | Автоматики и вычислительной техник |
| Кафедра | Автоматизированных систем управления |

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| по дисциплине | Системное программирование |
|  | |

|  |  |
| --- | --- |
| на тему | Турнир роботов. Разработка клиентского модуля |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ДАНО студенту | Ляскину Константину Алексеевичу | группы | КИ-18-01 |
|  | (фамилия, имя, отчество в дательном падеже) |  | (номер группы) |

**Содержание работы:**

|  |  |
| --- | --- |
|  | Введение |
|  | Основная часть |
|  | Заключение |

**Исходные данные для выполнения работы:**

|  |  |
| --- | --- |
|  | Правила проведения турнира |
|  | Спецификации ядра |
|  |  |

**Рекомендуемая литература:**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Графическая часть:**

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель: |  |  |  |  |  |  |  |
|  | (уч.степень) |  | (должность) |  | (подпись) |  | (фамилия, имя, отчество) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задание принял к исполнению: | студент |  |  |  | Ляскин К.А. |
|  | |  | (подпись) |  | (фамилия, имя, отчество) |

Оглавление

[Введение 4](#_Toc73514737)

[Правила турнира. 5](#_Toc73514738)

[Спецификации ядра. 6](#_Toc73514739)

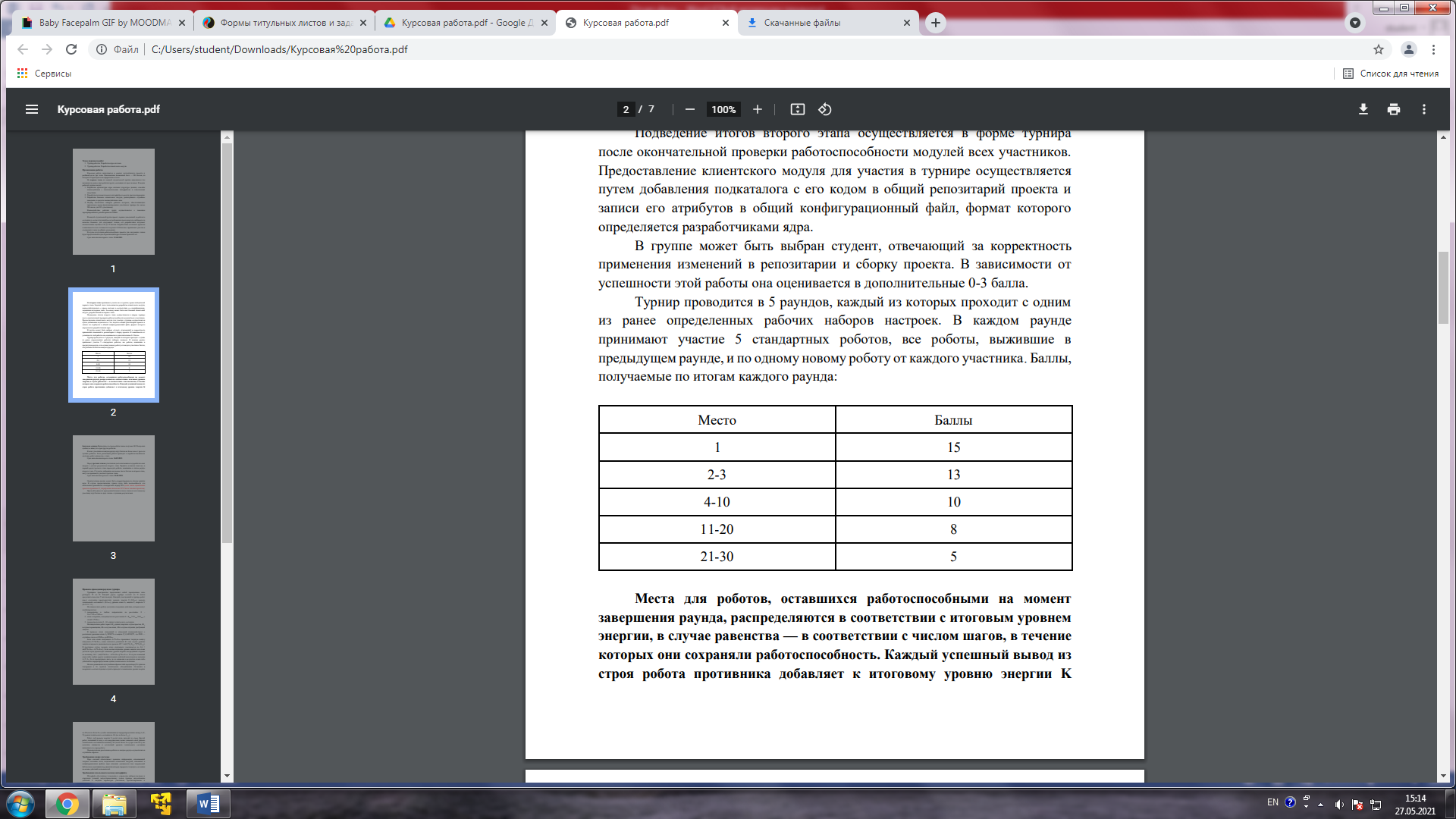
[Робот 8](#_Toc73514740)

# Введение

Разработка клиентского модуля, взаимодействующего с ядром системы в соответствии со спецификациями, заданными на первом этапе. За основу может быть взят базовый клиентский модуль, разработанный на первом этапе.

# Правила турнира.

Турнир проводится в 5 раундов, каждый из которых проходит с одним из ранее определенных рабочих наборов настроек. В каждом раунде принимают участие 5 стандартных роботов, все роботы, выжившие в предыдущем раунде, и по одному новому роботу от каждого участника. Баллы, получаемые по итогам каждого раунда:



Места для роботов, оставшихся работоспособными на момент

завершения раунда, распределяются в соответствии с итоговым уровнем

энергии, в случае равенства — в соответствии с числом шагов, в течение

которых они сохраняли работоспособность. Каждый успешный вывод из

строя робота противника добавляет к итоговому уровню энергии K

бонусных единиц. Вышедшие из строя роботы также получают K/2 бонусных

единиц за вывод из строя других роботов.

В зачет участника в каждом раунде идут баллы не более чем от трех его

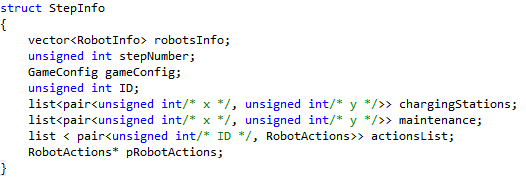
лучших роботов. Если реализация робота приводит к неработоспособности

системы, робот снимается с этапа.

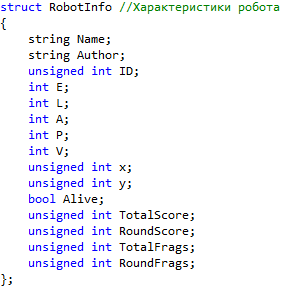
# Спецификации ядра.

Для добавления своего робота в общее решение, вам необходимо взять проект из папки Robots с именем RobotXX, где XX – номер по журналу (01, 02, …, 21). В основном cpp файле уже написана сигнатура функции DoStep с пустым телом. Все, что вам нужно сделать, это заполнить тело данной функции.

На каждом шаге роботу передается структура StepInfo:

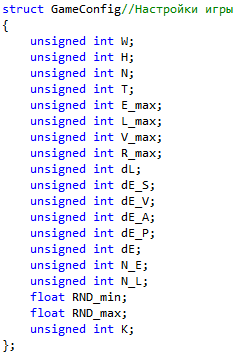


robotsInfo - информация обо всех роботах на текущем шаге, содержащая информацию следующего вида:



stepNumber – номер текущего шага;

gameConfig – настройки игры:



* ID – идентификационный номер робота
* chargingStations & maintenance – координаты станций подзарядки и технического обслуживания;
* actionsList – действия каждого робота на предыдущем шаге;
* pRobotActions – указатель на класс, хранящий действия вашего робота на текущем шаге;

# Робот

Создание функции передаваемой ядру игры

extern "C" \_\_declspec(dllexport) void DoStep(StepInfo \* \_stepInfo)

{

srand(time(NULL));

stepInfo = \_stepInfo;

int ID = stepInfo->ID;

В этом блоке мы считываем информацию о нашем роботе в объект структуры RobotInfo

for (auto it = stepInfo->robotsInfo.begin(); it != stepInfo->robotsInfo.end(); ++it)

{

if (\_stepInfo->ID == it->ID)

{

myInfo = new RobotInfo(\*it);

break;

}

}

Далее записываем полученные константные данные в созданные переменные и вычисляем значения по формулам:

maxDistance – максимальная дистанция которую может пройти робот.

maxAttackDistance – максимальная дистанция для про проведения атаки.

maxAttack – сила с которой робот бьет.

maxProtection – максимальный уровень защиты.

int maxDistance = Vmax \* V / Lmax \* E / Emax;

int maxAttackDistance = Rmax \* V / Lmax \* E / Emax;

int maxAttack = A \* E / Emax;

int maxProtection = P \* E / Emax;

Создаем два объекта структуры Point для обозначения расположения станций зарядки и ремонта.

Point energy, mech;

energy = getCStation(x, y);

mech = getMStation(x, y);

int S\_energy = Pythagoras(energy.x, energy.y);

int S\_merch = Pythagoras(mech.x, mech.y);

Здесь задается условие, что если у робота энергии меньше 700, или времени до конца раунда остается меньше 50 шагов, то робот уходит на станции подзарядки и перераспределяет характеристики. 50 уходит в атаку, 50 уходит в защиту.

if ((se == 0) && ((E < 0.7 \* Emax) || (stepInfo->stepNumber > 900)) && (L > 0.7 \* Lmax))

{

stepInfo->pRobotActions->addActionRedistribution(0, L, 0);

return;

}

Основная часть логики робота, здесь мы задаем и проверяем текущий уровень заряда и если он ниже критического уровня – то робот начнет поиск ближайшей станции зарядки. Если Заряда хватает для того чтобы добраться до энергостанции то направляемся к ней, иначе просто движемся в ее сторону. На станции робот перераспределяет характеристики.

if ((E < 0.7 \* Emax) || (stepInfo->stepNumber > 950))

{

if (maxDistToMove > S\_energy)

{

stepInfo->pRobotActions->addActionMove(energy.x, energy.y);

stepInfo->pRobotActions->addActionRedistribution(L \* 0.5, L \* 0.5, 0);

}

else

{

energy.x = energy.x \* maxDistToMove / S\_energy;

energy.y = energy.y \* maxDistToMove / S\_energy;

stepInfo->pRobotActions->addActionRedistribution(0, 0.3 \* L, 0.7 \* L);

stepInfo->pRobotActions->addActionMove(energy.x, energy.y);

}

return;

}

Если уровень энергии и технического состояния в пределах нормального, то считываем информацию о роботах и их местоположении.

VictimInfo getVictimInfo() {

UINT min\_dX = stepInfo->gameConfig.W + 1;

UINT min\_dY = stepInfo->gameConfig.H + 1;

int real\_dX = 0, real\_dY = 0;

UINT victimID = 1000;

for (auto it = stepInfo->robotsInfo.begin(); it != stepInfo->robotsInfo.end(); ++it)

{

Point vec = getDistance(Point(myInfo->x, myInfo->y), Point(it->x, it->y));

Далее проверяем, чтобы авторы других роботов не являлись одними из тех, кого мы не хотим бить.

if (!(it->Author == "Lyaskin" || it->Author == "Yeah" || it->Author == "Polyakov"))

{

real\_dX = vec.x;

real\_dY = vec.y;

min\_dX = abs(vec.x);

min\_dY = abs(vec.y);

victimID = it->ID;

}

Определяем случайные величины, с которыми робот может ударить или защититься от атак.

float A\_r = stepInfo->gameConfig.RND\_min \* A;

float P\_r = (1 - stepInfo->gameConfig.RND\_min) \* victim->P;

int delta = (A\_r \* E - P\_r \* victim->E) / stepInfo->gameConfig.E\_max;

В этом блоке кода, прописана логика атак.   
1. Вычисляем дистанцию атаки с учетом затрачиваемой энергии.

1. Вычисляем текущее расположение.
2. Проверяем расстояние, если его и энергии хватает для атаки, то атакуем.
3. В ином случае

if (delta > 0)

{

maxDistToAttack = stepInfo->gameConfig.R\_max \* myInfo->V \* myInfo->E / (stepInfo->gameConfig.L\_max \* stepInfo->gameConfig.E\_max);

if (myInfo->E - stepInfo->gameConfig.dE\_A > 0)

{

stepInfo->pRobotActions->addActionAttack(vInfo.ID);

}

Point newDist = getDistance(Point(myInfo->x, myInfo->y), Point(victim->x, victim->y));

if (Pythagoras(newDist.x, newDist.y) <= maxDistToAttack)

stepInfo->pRobotActions->addActionAttack(vInfo.ID);

}

Здесь прописываются желаемые характеристики для робота. Они должны быть распределены так, что в сумме не должны превышать больше 100.

stepInfo->pRobotActions->addActionRedistribution(0.45 \* L, 0.45 \* L, 0.1 \* L);

Схема движения робота определяется при помощи рандомайзера значение которого записывается в переменную direction. Далее в зависимости от значения direction в конструкции switch отправляется команда на передвижение.

int direction = (rand() + x) % 8;

switch (direction)

{

case 0:

stepInfo->pRobotActions->addActionMove(1, 0);

break;

case 1:

stepInfo->pRobotActions->addActionMove(-1, 0);

break;

case 2:

stepInfo->pRobotActions->addActionMove(0, 1);

break;

case 3:

stepInfo->pRobotActions->addActionMove(0, -1);

break;

case 4:

stepInfo->pRobotActions->addActionMove(1, 1);

break;

case 5:

stepInfo->pRobotActions->addActionMove(-1, 1);

break;

case 6:

stepInfo->pRobotActions->addActionMove(-1, -1);

break;

case 7:

stepInfo->pRobotActions->addActionMove(1, -1);

break;

default:

break;

}

return;

}

}