Общие сведения и требования ко всем заданиям семестра

В силе остаются требования, накладываемые на программы в прошлом семестре, причем некоторые из них ныне несколько ужесточаются. В частности:

- Обязательно использование утилиты make, gmake или cmake для автоматизации компиляции и сборки.
- Нужно придерживаться определенного стиля оформления кода (не должно быть никаких "магических чисел" – у констант должны быть имена, имена переменных и констант должны быть осмысленными, а не абстрактными и т.д.) Кроме того, крайне рекомендуется избегать без особой надобности глобальных переменных, а также оператора безусловного перехода goto (за исключением трех случаев, рассмотренных в файле «Примеры из семинарских занятий» на сайте cmcmsu.info), крайне аккуратно использовать setjmp, longjmp, особенно после изучения механизма исключений языка C++.
- Не забывайте про комментарии в коде.
- Программа должна быть выложена в соответствующий репозиторий на bitbucket.org, при этом должен быть написан файл README.md (некоторое описание языка markdown доступно здесь: http://ilfire.ru/kompyutery/shpargalka-po-sintaksisu-markdown-markdaun-so-vsemi-samymi-populyarnymi-tegami/) (на это будем обращать более пристальное внимание, включая снижение оценки):
    - Выбранный вариант задания (при наличии выбора) или описание особенностей вашей реализации, если выбора нет.
    - Процесс сборки проекта (наличие Makefile).
    - Процесс запуска проекта.
    - Описание, что в каких файлах с кодом находится (для многофайловых проектов).
    - Указание используемых технологий (в случае использования сторонних библиотек) с указанием источников. Использование сторонних библиотеки (кроме -lrt -lptread -ldl -lm) необходимо согласовывать с преподавателем.
    - Любая другая дополнительная информация, которую вы хотите довести до преподавателей, например, что какие-то функции не реализованы и т.п.
- Если ваша программа предполагает пользовательский интерфейс, то его дизайн должен соответствовать положительному пользовательскому опыту. Под пользовательским интерфейсом не обязательно понимается визуальная составляющая, командно-консольный вид – тоже пользовательский интерфейс, позаботьтесь о том, чтобы у вас были реализованы команды exit, help и т.п., кроме того сообщайте пользователю об ошибках, причем так, чтобы он мог их исправить (если это зависит от него), например сообщайте, что он ввел слишком длинный текст, который не может быть обработан вашей программой, – если вы этого не делаете, то предполагается, что ваша программа может обработать любой объем (например, частями). Проверку прочих ошибок (к примеру, невозможность выделить память) также никто не отменяет.
- Не забывайте проверять программу через средство valgrind.
- На усмотрение преподавателя вам может быть выдано дополнительное задание – как правило это мелкие доработки вашей текущей реализации, направленные на проверку общего понимания вашего кода.

По стилю оформления напоминаем про материалы А.В. Чернова:
https://ejudge.ru/study/3sem/style.shtml,
а также про книгу А.В. Столярова (http://www.stolyarov.info/books/codestyle), на основе рекомендаций из которой вы можете выработать свой стиль. По мнению А.Н. Сальникова использовать стиль ANSI и стиль snake_case в идентификаторах, но вы вправе придерживаться другого разумного стиля.

Варианты заданий будут разной сложности. Выбирайте вариант по своим силам. Не самый сильный студент, выполнив пусть даже не самое сложное задание качественно и в срок, может получить высокую оценку. И наоборот, сильный студент, выбирая простейший вариант задания, не может рассчитывать на высокую оценку.
Каждый Task в этом семестре будет оцениваться **максимально в 20 баллов** (если не сказано иного), при этом у вас появляется возможность **заработать несколько дополнительных баллов** (на

усмотрение преподавателей) при реализации большего функционала и/или качественном подходе к выполнению задач.

Task1 – общие сведения

Цель – освоить клиент-серверную модель взаимодействия процессов, основанную на сокетах.
Средство – языки Си или C++ (для тех, кто уже знаком с языком C++).

Базовая модель

- Процесс-сервер:
  - создает "слушающий" сокет и ждет запросов на соединение от клиентов;
  - приняв запрос – создает сыновний процесс/поток, который должен обслужить клиента и завершиться, а процесс-отец продолжает принимать запросы на соединение и создавать новых сыновей/потоки.
- Процесс-сын (обслуживание клиента) – получить от процесса-клиента данные, обработать их и вернуть клиенту, после чего завершиться – возможна обработка нескольких порций данных от клиента вплоть до получения от клиента команды на завершение.
- Процесс-клиент запрашивает у пользователя данные, отправляет их серверу (адрес сервера можно задать в командной строке, или спросить у пользователя), получает от сервера ответ, выполняет преобразования (или некие действия, например, печать на экран).

Такая модель предполагает, что клиенты не общаются между собой через сервер, а взаимодействуют только с сервером (каждого клиента обслуживает отдельный серверный процесс). Реализацию подобной модели можно посмотреть на сайте http://cmcmsu.info/, раздел "второй курс, весенний семестр" в методическом пособии "Модельный SQL-интерпретатор. (2005 г.)".

Модель с межклиентским взаимодействием

- Процесс-сервер:
  - создает "слушающий" сокет и ждет запросов на соединение от клиентов;
  - в бесконечном цикле – либо принимает запрос на подключение от клиента, либо обрабатывает данные полученные от клиента. Обработка данных может включать в себя в том числе отправку данных конкретному клиенту, отправку данных всем клиентам, сохранение данных и т.п.
- Процесс-клиент запрашивает у пользователя данные, отправляет их серверу (адрес сервера и номер порта задать в командной строке (аргументах main)), получает от сервера ответ, выполняет преобразования (или некие действия, например, печать на экран).

Очевидно, что в данной модели сервер выступает в качестве управляющего узла, ему необходимо знать всех подключенных клиентов (а не только одного как это делает процесс-сын в базовой модели) и уметь понимать, чего эти клиенты хотят. В этом помогут системные функции select() или более современная его версия – poll(), смотрите описание в справочнике man.

[Системный вызов select() подробно описывается в методическом пособии А.В. Столярова "Многопользовательский игровой сервер" (http://www.stolyarov.info/) или в разделе "Учебные пособия" кафедрального сайта АЯ (http://al.cs.msu.su/), также в книге А.В. Столярова "Программирование", том 3 (стр. 222 в http://www.stolyarov.info/books/pdf/progintro_vol3.pdf). Кроме того, достаточно подробное описание этой функции и других для работы с сокетами на языке С можно найти здесь https://rsdn.org/article/unix/sockets.xml и здесь http://citforum.ru/programming/unix/sockets/, а для тех, кого не пугает английский язык, есть еще неплохая презентация https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf .]

Для тестирования как базовой модели, так и расширенной модели не обязательно иметь несколько компьютеров – можно запускать клиентов из разных консолей на одном компьютере, хотя при желании вы можете развернуть сервер где-нибудь в сети (например, на digitalocean.com) и даже подружить ваши программы клиенты с чужими программами-серверами.

Дополнительные требования и условия к Task1

- Не забывайте про обработку сигналов как на сервере, так и в клиенте (нажатие клавиш ctrl+C посылает сигнал SIGINT, ctrl+Z посылает SIGSTP, команда kill <pid_процесса> посылает SIGTERM). При приостановке сервера по ctrl+Z может разорваться tcp-соединение (но сокеты останутся занятыми), и при перезапуске из «замороженного состояния» (по сигналу SIGCONT) соединение не будет работать.
- Обращайте внимание на точность ввода команд в ваших программах, если у вас реализована команда \help, то команда \helpppp не должна восприниматься как команда.
- Пустое сообщение – состоящее только из пробельных символов, не является сообщением, так же как и пустой ник (в случае чата) не является корректным.
- Длинные сообщения (превышающие размер буфера) должны обрабатываться одним из трех способов:
    1. Вывод сообщения об ошибке.
    2. Передача сообщения по частям и склейка частей в исходное состояние.
    3. “Правильное” разбиение текста (например, по пробелу, слова не должны разбиваться пополам), передача частями, вывод отдельных частей без склейки, причем так, чтобы сообщения других пользователей не вклинивались в части большого сообщения.
- **Во всех вариантах** при запуске сервера указывается номер порта, на котором сервер будет ожидать подключения клиентов. При запуске клиента указывается имя компьютера (адрес) и номер порта для подсоединения к серверу. В файле readme должен быть указан точный текстовый протокол запуска и взаимодействия клиента и сервера.
- Общие для всех вариантов правила форматирования сообщений:
    - приватное сообщение начинается с *;
    - оповещение о событии на канале, начинается с *** (например, пришел или ушел пользователь);
    - служебное сообщение начинается с ### (например, завершение работы сервера).
- Сообщения, отправляемые программой-клиентом, могут быть двух типов:
    - команда, начинается с \ и имеет следующий вид \<команда> <параметр> ... <параметр>, при этом лишние пробельные символы между параметрами игнорируются; **если команда не распознана, такое сообщение считаем обычным**;
    - обычное сообщение (реплика) чата.
- Клиент может завершать свои строки любой из комбинаций: \n, \r, \n\r, \r\n (сервер поддерживает разные типы клиентских программ).
- Сервер начинает обработку сообщения от клиента, когда получит символ(ы) конца строки. До этого момента сообщение (строка) накапливается во временном буфере. Максимальная длина одной строки, получаемой сервером, может быть как фиксированной, так и динамической (помните про обработку длинных сообщений).
- Начальные и завершающие пробельные символы (пробел, табуляция) игнорируются (т.е., строка " hi ppl " будет обработана так же, как "hi ppl").
- Когда сервер завершает свою работу, он должен прежде попрощаться со всеми клиентами, например так: «### server is shutting down, thanks to everyone».

Для организации взаимодействия процессов необходимо использовать сокеты коммуникационного домена AF_INET с установлением соединения (SOCK_STREAM).

Студенты, выполняющие одинаковый вариант, могут объединяться в "консорциумы" для выработки единых стандартов (правил) общения сервера и клиента. Если придерживаться выработанных правил, клиент одного студента, сможет общаться с сервером другого студента и наоборот. Будет происходить взаимное тестирование.

Заметим, что "консорциум" не означает командное программирование, каждый студент выполняет задание самостоятельно, зная только "правила взаимодействия" клиентов и серверов, и не имея представления о внутреннем устройстве чужой программы.
Такое взаимодействие возможно не только для чатов, но и для "простого" задания. Например, сервер одного студента увеличивает число на 1, другого – на 5, третьего – на 10. Тогда клиент может воспользоваться услугами разных серверов, в зависимости от того, на сколько нужно увеличить число. Если на 12, то один раз нужно обраться к третьему серверу и два раза – к первому.

Приведенный ниже набор вариантов является переработкой заданий, предлагаемых в разные годы Ю.С. Коруховой, П.Г. Сутыриным, А.А. Вылитком. Текст заданий дополнен К.М. Ивановым.
Задание игра «Консольный DOOM» предложено А.Н. Сальниковым.

Варианты задания (сложность указана звездочками):

Вариант 1 (*) – Калькулятор 1.
Процесс-сервер – создает "слушающий" сокет и ждет запросов на соединение от клиентов. Приняв запрос, процесс-сервер создает сыновний процесс, который должен обслужить клиента и завершиться, а процесс-отец продолжает принимать запросы на соединение и создавать новых сыновей.

Задача сына (обслуживание) – выполнять возможные команды от клиента:

1) \+ <число> – установить число на которое сервер будет увеличивать числа для данного клиента (по умолчанию 1), вернуть клиенту “Ok”;
2) <число> – запрос на увеличение числа от клиента, задача – увеличить данное число на заданное и вернуть клиенту;
3) \? – получить от сервера число, на которое тот увеличивает числа для текущего клиента;
4) \- – сообщить серверу о завершении работы, при этом сервер-сын завершается.

Программа-клиент:

1) запрашивает у пользователя очередную команду, отправляет ее серверу (адрес сервера можно задать в командной строке (передаются в main() через argv), или спросить у пользователя первым действием;
2) получает от сервера ответ и печатает его на экран.

В качестве программы клиента возможно использование утилит telnet или netcat (в разных системах может называться nc, netcat, ncat, pnetcat, не входит в стандарт POSIX).

Вариант 2 (**) – Калькулятор 2.
Клиентам доступны те же команды, что и в Варианте 1, с некоторыми изменениями:

1. Все клиенты обслуживаются единым сервером (а не отдельными сыновьями).
2. Команда +<число> устанавливает глобальное для всех клиентов число, на которое будет увеличен любой запрос вида <число>.
3. Сервер ждет пока к нему не придет два числа от одного и того же или нескольких клиентов (с помощью команды <число>) складывает эти два числа, прибавляет к ним то, что установлено глобально и возвращает только тем клиентам, которые участвовали в операции (прислали эти два числа).

В данном варианте и в дальнейших сервер завершает свою работу только в случае ввода (если не указано иного):

- команды \exit на самом сервере (от клиентов такая команда не принимается)
- сигнала SIGINT (по нажатию ctrl+C на клавиатуре)
- сигнала SIGSTP (по нажатию ctrl+Z на клавиатуре)
- сигнала SIGTERM (команда kill <pid_процесса>).

Вариант 3 (**) – Чат 1.

Процесс-сервер:

1) создает "слушающий" сокет и ждет запросов на соединение от клиентов;
2) при поступлении запроса, устанавливается соединение с очередным клиентом, от клиента сервер получает имя (ник) вошедшего в "комнату для разговоров", клиент заносится в список присутствующих;
3) всем присутствующим рассылается сообщение, что в комнату вошел такой-то (имя);
4) от разных клиентов могут поступать реплики – получив реплику от клиента, сервер рассылает ее всем "присутствующим" (включая самого автора) с указанием автора реплики;
5) при разрыве связи (команда \quit) с клиентом сервер сообщает всем, что-такой-то (имя) нас покинул (ушел) и выводит его прощальное сообщение.

Необходима реализация команд:
а) \users получить от сервера список всех пользователей (имена), которые сейчас онлайн;
б) \quit <message> – выход из чата с прощальным сообщением.
В качестве программы клиента возможно использование telnet. Имена пользователям могут выдаваться автоматически, но они должны быть уникальными. Иной подход – сервер запрашивает имя клиента.

Вариант 4 (***) – Чат 2.
Условия аналогичны Варианту 2 с дополнительными командами:

1) \private <nickname> <message> приватное сообщение пользователю с именем <nickname>, если пользователя-адресата на сервере нет, то выдается сообщение об ошибке;
2) \privates – получить от сервера имена пользователей которым вы отправляли приватные сообщения;
3) \help – вывод допустимых команд.

При запуске программы-клиента указывается имя компьютера (адрес) и номер порта для подсоединения к серверу, а также имя пользователя – проверка на уникальность имени обязательна.

В данном и всех последующих вариантах telnet может выступать только в качестве дополнительного средства взаимодействия с сервером – наличие программы клиента обязательно.

Вариант 5 (***) – Автосалон.

Программа-сервер имитирует деятельность автосалона, который продает и покупает подержанные автомобили у программ-клиентов.
Программа-клиент ожидает ввода сообщений со стандартного потока ввода и одновременно ожидает поступления сообщения от сервера:
- При поступлении сообщения от сервера оно печатается на стандартный поток вывода.
- При вводе сообщения с клавиатуры оно пересылается на сервер.

Со стандартного потока ввода читаются сообщения в следующем формате:
- BUY <марка машины>
    - Когда сервер получает очередное сообщение BUY … от какого-либо клиента, он удаляет автомобиль из своей базы, если такой автомобиль есть, и рассылает всем клиентам сообщение о том, что машина куплена. Если таких машин несколько, то удаляется первая.

    - Если машина успешно продана, то продавец и покупатель получают сообщения с уведомлением о сделке и поздравлениями.
    - Если запрошенной машины нет – отправителю BUY сервер сообщает об ошибке.
- SELL <марка машины>
    - При получении запроса SELL … сервер добавляет машину к своей базе данных и рассылает всем клиентам сообщение о появлении нового автомобиля в салоне

При попытке ввода сообщений, не соответствующих формату, программа-клиент должна сообщать об ошибке.

Вариант 6 (****) – Чат 3.

Расширение Варианта 4:
Появляется новый тип пользователей – администраторы, для того чтобы получить доступ к функционалу администраторов пользователь должен выполнить команду \admin, после чего сервер запрашивает пароль, при вводе корректного пароля пользователь получает доступ к дополнительным командам:

- \ban <nickname> <message> – отключает текущего пользователя с заданным именем и запрещает новые подключения к серверу с таким именем, при этом пользователю выводится причина его блокировки, указанная в <message>;
- \kick <nickname> <message> – отключает пользователя с заданным именем, при этом пользователю выводится причина его блокировки, указанная в <message> ;
- \nick <oldnickname> <newnickname> – принудительно меняет имя пользователю;
- \shutdown <meassage> – завершает работу сервера, а всем пользователям отправляется заданное сообщение.

Администраторы не могут влиять на других администраторов.
Обычные пользователи также получают возможность менять свое имя командой \nick <новое_имя>, если такой пользователь уже есть, должно выдаваться сообщение об ошибке тому, кто пытается занять это имя. Можно выдавать сообщение и тому клиенту, чьё имя пытаются занять, с указанием IP-адреса или имени второго клиента.

Вариант 7 (*****) – Чат 4.

Расширение варианта 6, вводится поддержка нескольких каналов:

- Прежде чем начать писать сообщения клиент должен выбрать канал (комнату) командой /room <название_канала> и подключиться к каналу, если он есть. Если пользователь на этом канале заблокирован, выдается сообщение, содержащие причину блокировки. Если канала нет, то таковой создается, и пользователь становится его создателем, а также его **главным администратором** (при этом сервер запрашивает пароль, который будут использоваться для всех администраторов канала).
- Все функции администраторов, описанные в Варианте 5, теперь действуют в рамках канала (кроме \shutdown – она более недоступна простым администраторам).
- Команда \users без параметров действует в рамках канала, команда \users <chanel> позволяет узнать список пользователей канала.
- Администраторам канала становится доступна новая команда:
    - \topic <newname> – позволяет менять имя канала.
- Главному администратору становится доступны команды:
    - \setadmin <nickname> – высылает заданному пользователю пароль от функций администратора;
    - \banadmin <nickname> – запрещает пользователю использовать функционал администратора, даже если тот имеет пароль;
    - \unbanadmin <nickname> – разрешает пользователю быть администратором;

    - \delete – удаляет канал;
    - \passwd <password> – позволяет изменить пароль администратора, при этом он высылается всем не заблокированным администраторам.
- Обычным пользователям становится доступна команда \leave, позволяющая покинуть канал.
- Команды \private и \privates продолжают действовать в рамках всей системы.
- Список доступных каналов можно узнать командой \rooms.
- Вводится новый тип пользователя server admin (\serveradmin) – не может воздействовать на каналы, но может завершить работу сервера (\shutdown), а также перезагрузить его (\restart <message>). Сервер должен завершать и начинать свою работу так, чтобы между запусками сервера сохранялась следующая информация (соответственно нужно придумать формат файла, читать и писать в него по необходимости):
    - имена, пароли и статусы (создатель, оператор) всех пользователей;
    - каналы, их темы, списки заблокированных пользователей с причинами;
    - отложенные сообщения (если реализованы).
- Поддержка так называемых отложенных (offline) сообщений. Если адресатом в команде \private указан пользователь, зарегистрированный на сервере, но не подключенный в данный момент, то сообщение ставится в очередь (не более 10 для одного получателя), а отправителю посылается объяснение ("user is offline right now, your message will be delivered upon his/her connection"). Соответственно, при подключении адресата ему первым делом доставляются личные сообщения из очереди. В этом случае адресат должен видеть, когда они были отправлены (абсолютное время или относительное: "5 hours 45 minutes 3 seconds ago").

В данном варианте допустима частичная реализация функционала, при этом в README.md должно быть четко зафиксировано, что реализовано, а что нет.

Вариант 8 (******) – GUI Чат.

Расширение Варианта 7 – визуальный пользовательский интерфейс для чата, что можно реализовать:

- одновременное присутствие на нескольких каналах (при желании каждый канал можно реализовать на отдельном сервере-сыне);
- одновременное ведение нескольких приватных бесед (беседа как последовательность приватных сообщений, начинается первым сообщением, фактически является каналом без администраторов);
- отдельное окно для набора своего сообщения, не прерываемого потоком канала ;
- отдельное окно для списка пользователей на канале;
- и т.д.

Для реализации оконного клиента можно воспользоваться библиотекой ncurses (существуют различные версии этой библиотеки).

Для заинтересованных студентов: можно воспользоваться qt на языке C++ или MFC (по согласованию с преподавателем подходящей среды программирования) и аналогами.

Вариант 9 (*****) – Сетевая игра: консольный Doom

Требуется реализовать игровой сервер, игровой клиент и монитор проводимых игр. Сервер и клиенты должны взаимодействовать через сеть путём установки TCP соединения.

При запуске сервер читает файл с картой (имя файла указывается в параметрах при запуске). Сервер должен быть реализован как демон в Unix, тем самым вся диагностическая информация должна выдаваться в специальный файл журнала (можно в syslog). Для имени файла карты и файла журнала в программе должны быть предусмотрены имена по умолчанию, которые будут подставлены, если имена файлов не были указаны в командной строке при запуске. Также специальными параметрами указываются номер TCP порта и имя файла, где будет записан pid сервера (это необходимо чтобы можно было посылать сигналы именно этому экземпляру сервера), по умолчанию: /var/run/название_моего_сервера). Также необходимо иметь возможность запустить сервер не как демон, а как обычную программу.

Клиент соединятся с сервером (hostname сервера и номер TCP порта сервера указываются клиенту в аргументах main при запуске (если номер порта не указан, использовать некоторый номер порта по умолчанию). Клиент должен отображать игровое пространство, позволять игроку делать ход, выходить из игры по желанию игрока (в том числе корректно завершаться по нажатию на Ctrl+c и Ctrl+d).

Монитор игр – это отдельная программа, которая может обращаться к серверу с применением IPC средств и показывать сведения о ведущихся на сервере играх.

Игроки на сервере могут выступать в 2-х ролях: в роли создателя команды игроков и в роли участника команды игроков. В начальный момент, перед тем как начать игру, одним из игроков создаётся команда, и он ожидает, пока нужное количество других участников присоединится к этой команде, чтобы начать игру. Именно создатель команды определяет число участников игры и момент начала игры. При подключении клиент может просмотреть список уже имеющихся команд. При подсоединении или создании команды игрок создаёт себе <<имя>> login. Размер имени не может превышать 60 символов.

Описание игры

Игра начинается после того, как создатель команды игроков объявил старт игры. Если игра началась, то присоединиться к ней ещё одному игроку нельзя до тех пор, пока она не будет закончена. Создатель команды игроков может по своему желанию в любой момент завершить игру. Создатель не может принимать участия в игре, но зато по запросу может узнать координаты игроков и их уровень здоровья.

Игра происходит в прямоугольном лабиринте, представленным как набор точек некоторой матрицы размера M×N. Лабиринт вместе с его размером должны быть заданы в файле карты, при этом сам файл карты должен иметь текстовое представление, легко читаемое человеком. Цель игры – оказаться выжившим в лабиринте с максимальным уровнем здоровья. Лабиринт состоит из стенок, аптечек, и проходов. По точке, содержащей аптечку, можно двигаться, по стенкам и за границами лабиринта двигаться нельзя.

Аптечка обладает либо лечебным, либо отравляющим эффектом определённой силы. Игрок может съесть аптечку, при этом к его здоровью либо прибавляется, либо отнимается численное значение эффекта. По внешнему виду аптечки ничего нельзя сказать о силе её воздействия и о знаке её воздействия (лечить или отравляет). После употребления аптечки соответствующая клетка считается просто проходом (повторно съесть аптечку нельзя). В лабиринте могут встречаться

другие игроки, которые всегда занимают одну точку пространства (в стенке игрок не может находиться).

Игроку сопоставляется некоторый уровень здоровья, который с каждым сделанным им ходом уменьшается на определённое значение (указывается в параметрах серверу). С течением времени, если игрок не перемещается, значение здоровья уменьшается, но не так активно, как в случае перемещения. Игрок помещается в одну из точек с координатами (i, j), два игрока не могут одновременно находится в одной и той же точке. В этой точке он может видеть состояние лабиринта на 10 точек в каждом направлении. То есть будет виден прямоугольник с координатами (i-10, j-10, i+10, j+10). Прямоугольник и всё что в нём есть нужно уметь показывать в консоли в текстовом режиме.

За один ход можно произвести одну из следующих операций:

- съесть аптечку, находящуюся в текущей точке,
- применить боевой заряд,
- переместиться на 1 соседнюю клетку,
- заложить мину.

Боевой заряд применяется следующим образом. Вычисляется кратчайшее расстояние, до другого игрока по пути через клетки (если игрок оказался за стенкой, то стенки надо обходить). Радиус действия заряда не превышает 10. Интенсивность удара убывает с расстоянием от игрока, который заряд применяет. Игрока, применившего заряд, удар от этого заряда не травмирует.

Игроку предоставляются мины в количестве 10 штук, каждой из которых он может заминировать клетку в лабиринте. Мины не отображаются другим игрокам. В случае попадания игрока на мину к нему применяется ущерб, эквивалентный применению боевого заряда на соседней клетке.

В случае применения заряда есть время, необходимое на перезарядку, в случае минирования игрок на определённое время обездвиживается.

В начальный момент игроки расставляются сервером на карту случайным образом. Они должны обязательно оказаться в допустимой точке (то есть не на стенке и не на одной клетке с другим игроком). С начала игры объявляется определённой длительности мораторий на применение оружия. Длительность моратория задаётся в файле с картой и измеряется в секундах.

Мощность заряда и скорость убывания здоровья при движении может быть задана в файле с картой. Значения действий аптечек также указываются в файле с картой.

Формат файла с картой

Нижен приведён пример файла с картой:

```
Map 10x20

####################

# #  #   #          #

# #   #  #  #####    #

# #####  #      #    #

#        ######### ###
```

```
#  # ##      #       #
#  ########  # #####  #
## ##        # #     #
## ### ############ #
#  #        #     #  #
#       #      #     #
######################
```

```
initial_health       = 500
hit_value            = 50
recharge_duration    = 3
mining_time          = 6
stay_health_drop     = 1
movement_health_drop = 4
step_standard_delay  = 0.1
moratory_duration    = 5
```

```
items:
1  1     10
20 20   -10
2  3     8
```

В файле карты, собственно карта обрамлена контуром из решёток, что делает ее более наглядной. Координаты в карте нумеруются начиная с единицы.

Монитор игр и файл журнала

При обращении к монитору (отдельная программа) должен выдаваться список команд игроков, где для каждой команды выдаётся:

- статус игры: идёт, окончилась, не начата;
- дата начала игры, если начата;
- дата конца игры, если закончилась;
- победитель, если определён;
- список игроков, их текущий уровень здоровья и координаты.

В файл журнала нужно сохранять сведения о начавшихся и закончившихся играх, а так же выводить имена хостов подключившихся клиентов. Если клиент отключился, с использованием соответствующей команды, отправляемой серверу, то факт отключения клиента также нужно зафиксировать в журнале.

Сервер, в случае достаточности ресурсов, должен обеспечивать возможность подключения до 1000 клиентов одновременно.