Для **быстрого знакомства с материалами** смотрите раздел 6.

Стрелялка

Предлагается написать программу управления летательным аппаратом для простой однопользовательской игры. Каждый участник разрабатывает свою программу и набирает очки в игре независимо от остальных участников.

Жанр игры: **«вертикальный скролл-шутер»**. Имеется квадратное игровое поле, на котором расположен самолёт *игрока*, а также появляются разные летающие *враги*. Игрок и враги могут стрелять, в результате чего по полю также летают *снаряды*. Игрок может стрелять только вверх, при этом большая часть врагов движется вниз. Столкновение с чужим снарядом наносит урон как игроку, так и врагу. Игрок может таранить врагов: при этом ему также наносится урон.

Игроку даются *очки* за уничтожение врагов и за прохождение *уровней*. Большую часть врагов уничтожать не обязательно: ушедшие за нижную границу поля враги просто пропадают (без начисления очков). Обычно больше очков сулит прохождение в более поздние уровни. Некоторых врагов требуется непременно победить. Чтобы игра не могла длиться бесконечно, по истечении отведённого игрового времени такие враги могут сильно разозлиться.

В силу необычности первой номинации, рекомендуется задавать вопросы. Жюри постарается ответить на все вопросы, на которые оно в состоянии ответить.

1 Моделирование игры

На игровом поле введены координаты X и Y в пределах от -1000 до 1000. Координата X направлена вправо, а координата Y направлена вверх. Геометрически все объекты в игре представляются кругами. Снаряды при вылете за пределы поля пропадают. Игрок не может вылетать за пределы игрового поля. Враги также не могут вылетать за пределы поля, но при пересечении нижней границы большинство врагов пропадает.

Есть два вида времени: *игровое время*, по которому моделируется игра, летают снаряды, создаются новые враги и так далее; и астрономическое время, которое можно измерить по своим наручным часам. Игра моделируется по *тактам*, между соседними тактами всегда проходит ровно 1/60 секунды игрового времени. Астрономическое время обычно идёт примерно с той же скоростью, что и игровое, хотя возможны небольшие отклонения. Решение может узнать номер текущего такта и, разделив его на 60, получить текущее игровое время.

Игровой сервер моделирует игру в режиме реального времени, параллельно с программой игрока. Программа игрока может периодически посылать новые указания по контролю своего самолёта, а в обмен получать новую информацию об игровом мире. Если программа не успевает отправить новые указания до очередного такта, то игровой сервер продолжает применять последние полученные инструкции.

Размещение игровых объектов определяется генератором псевдослучайных чисел, который инициализируется от некоторого значения `seed` и ведёт себя детерминированно. На сервере жюри `seed` задаётся специальным образом, а локально его можно установить с помощью ключа `--seed`. Обратите внимание, что из-за сложностей синхронизации игрового сервера и программы участника даже запуски игры с одинаковым `seed` могут приводить к разным результатам.

2 Языки программирования

На первой номинации поддерживаются языки программирования и компиляторы:

- Visual C++
- MinGW C++
- Java
- Python 3

Кроме того, предоставляются готовые проектные файлы для Visual C++, IntelliJ IDEA и PyCharm.

Для каждого из языков в архиве материалов есть пример простейшего решения. В коде примера есть часть, которая отвечает за подключение к игре: она состоит из вызова инициализации, а также цикла, внутри которого вызывается функция `Think` и обмен данными с игровым сервером. Изменять эту часть не рекомендуется. Также есть секция, отделённая горизонтальными линиями, в которой определена функция Think — эту часть программы вам требуется изменять. Эта функция запускается регулярно: она должна смотреть на состояние игрового поля `world`, и, исходя из этого, устанавливать контроль игрока `control`.

Запрещается открывать какие-либо файлы, вызывать функции операционной системы (WinAPI), создавать потоки, процессы, пытаться помешать нормальной работе игрового сервера и проверяющей системы.

3 Игровой API

Игровой мир записывается в типе данных `WorldData`, в котором хранится по одному массиву для объектов каждого типа. У каждого живого объекта можно указать:

1. номер слота — это номер ячейки в соответствующем массиве `WorldData`,
2. идентификатор `id` — некоторый уникальный номер объекта.

Оба значения не изменяются, пока объект жив. Каждому создаваемому объекту выдаётся следующий по порядку идентификатор, поэтому идентификатор отличается даже для объектов разного типа, и даже для объектов, переиспользующих один и тот же слот.

Все массивы в `WorldData` имеют фиксированный размер, но количество объектов постоянно изменяется в ходе игры. Это достигается за счёт того, что в массивах есть «мёртвые» ячейки. У этих ячеек поле `id` равно -1: следует полностью игнорировать такие ячейки.

В таблице описаны типы и поля:

`PlayerControl`	определяет контроль игрока
`.velocity`	относительный вектор скорости (умножается на 400 — максимальную скорость игрока)
`.shoot`	нужно ли сейчас стрелять
`.message`	произвольный текст длиной не более 256 символов (показывается около игрока в визуализаторе)

`PlayerData` или `EnemyData` или `ProjectileData`	ниже описана общая информация для игрока, врага и снаряда
`.id`	уникальный целочисленный идентификатор объекта (значение -1 означает, что этого объекта нет)
`.position`	координаты центра объекта
`.radius`	радиус объекта
`.velocity`	абсолютный вектор скорости объекта
`.health`	сколько жизней осталось у объекта (для снаряда означает наносимый урон)
`.maxHealth`	максимально возможное значение `health`

`EnemyData`	дополнительная информация о враге
`.type`	тип врага: определяет параметры, поведение, жизни, очки и т.п.
`.collisionDamage`	урон игроку при таране этого врага

`ProjectileData`	дополнительная информация о снаряде
`.type`	тип снаряда: определяет параметры, урон, скорость и т.п.
`.owner`	кто выпустил этот снаряд (`id` объекта-хозяина)

`WorldData`	полная информация об игровом мире
`.players`	массив игроков
`.enemies`	массив врагов
`.projectiles`	массив снарядов
`.tickCount`	номер такта игры (если разделить на 60, то получится игровое время)

На языках C++/Python можно работать с полями напрямую в стиле:

```
int x = world->enemies[i].position.x;        (C++)
int x = world.enemies[i].position.x          (Python)
```

На языке Java нужно вызывать getter/setter-функции в стиле:

```
int x = world.getEnemy(i).getPosition().getX(17)        (Java)
```

Кроме того, в программе можно использовать некоторые константы (см. `Swig_Constants.h`). В основном, это параметры игрока: радиус, скорость, начальные жизни, скорость стрельбы, урон, и т.д.

Решению участника доступны функции API:

- `PlayerApi_Init()` — инициализация и связь с игровым сервером;
- `PlayerApi_Update(control)` — отослать на игровой сервер заданный контроль, и обновить данные об игровом мире `world`;
- `PlayerApi_Log(sev, message)` — записать в текстовый лог `player.log` заданное строковое сообщение;

Первые две функции уже используются в примерах решений, вызывать их самостоятельно не рекомендуется. Третья функция может быть использована для отладки и диагностирования проблем, в том числе, при тестировании на сервере жюри. Уровень `sev` определяет, куда писать лог: 2 — писать только в файл, 3 — писать в файл и в консоль, 4 — важное сообщение в файл и в консоль. Уровни выше 4 обозначают ошибку и автоматически завершают вашу программу, так что их использовать не рекомендуется.

В решениях на C++ рекомендуется использовать функцию `check` вместо `assert`: она пишет сообщение об ошибке в лог и выбрасывает исключение.

Подробнее особенности синтаксиса каждого языка можно посмотреть в примере решения.

4 Ограничения

Ограничение по памяти составляет 256 мегабайт. При превышении этого ограничения игра будет немедленно завершена с сохранением уже набранных очков. Ограничения по времени нет: если ваше решение работает медленно, то оно сможет реже видеть состояние игрового поля и осуществлять контроль.

5 Отправка решений и система оценки

Очки даются за уничтожение врагов и за прохождение уровней. Чем больше очков набирает решение участника, тем лучше для него. Текущие очки показываются в правом верхнем углу визуализатора после слова `Score`.

Участники могут отправлять свои решения в систему тестирования NSUts. Во время тура решение запускается вместе с игровым сервером с фиксированным значением `seed`, которое равно 12345. Если вы отправили новое решение, когда старое ещё не было протестировано, то старое решение удаляется из очереди. Историю и текстовые логи своих игр вы можете скачать и посмотреть (подробнее в разделе 11). В течение тура вам доступен текущий предварительный рейтинг, результаты которого **не** идут в зачёт.

После конца тура будет выполнено **финальное тестирование**. От каждого участника в финальное тестирования проходит одно решение — **последнее успешно скомпилировавшееся решение**, отправленное в систему NSUts. Жюри выбрало для финального тестирования десять разных значений `seed`. Каждое решение будет запущено по одному разу с каждым из десяти этих значений. Три наихудших результата будут отброшены, а сумма очков за семь оставшихся запусков определяет результат первой номинации.

6 Материалы

Архив с материалами можно скачать по ссылке на вкладке «Новости» в системе NSUts. Для работы рекомендуется использовать Far Manager или командную строку Windows. Предполагается, что весь архив будет распакован в некоторую директорию, и вся работа будет вестись в ней.

Шаги для быстрого знакомства с материалами:

1. Откройте прилагаемый проект/солюшен для IDE вашего языка программирования.
2. Запустите отладку. Если визуализатор не показывается, запустите также игровой сервер: `NsuGameServer.exe -d`
3. Посмотрите меню визуализатора, попробуйте ручное управление, посмотрите на сообщения в текстовых консолях.
4. Попробуйте поставить breakpoint в решении и убедитесь, что отладка работает.
5. После завершения игры посмотрите выходные файлы игры (описано в разделе 10).
6. Запустите `NsuGameServer.exe current.replay`, чтобы посмотреть историю вашей игры.
7. Попробуйте из командной строки собрать и запустить ваше решение (см. разделы 7 и 8).
8. Отправьте пример решения в систему тестирования nsuts. Скачайте логи из nsuts и просмотрите историю игры (раздел 11).

Полное содержимое архива:

- `NsuGameServer.exe` — игровой сервер и визуализатор (подробности в разделах 8 и 12)
- `SamplePlayerCpp.cpp` — пример решения на C++
- `*.h`, `PlayerApiCpp.*` — служебные файлы для решений на C++
- `VCSolution/VCSolution.sln` — солюшен для Visual C++
- `SamplePlayerJava.java` — пример решения на Java
- `*.java`, `PlayerApiJava.dll` — служебные файлы для решений на Java
- `IdeaProject/` — директория с проектом IntelliJ IDEA
- `SamplePlayerPython.py` — пример решения на Python
- `PlayerApiPython.py`, `_PlayerApiPython.pyd` — служебные файлы для решений на Python
- `PyCharmProject/` — директория с проектом PyCharm

7 Сборка решений

При сборке решения участника следует правильно подключить библиотеку для связи с сервером. Ниже описан простой способ собрать прилагаемый пример решения в командной строке.

Visual C++: `cl /O2 SamplePlayerCpp.cpp PlayerApiCpp.lib`
В текущей директории при сборке должны находиться файлы с расширением .h , а при запуске — файл PlayerApiCpp.dll. В систему тестирования следует отправлять один файл SamplePlayerCpp.cpp

MinGW C++: `g++ -m32 -O2 SamplePlayerCpp.cpp PlayerApiCpp.dll -o SamplePlayerCpp.exe`
В плане предназначения файлов всё аналогично варианту Visual C++.

Java: `javac *.java` *или* `javac SamplePlayerJava.java`
В текущей директории при сборке должны находиться файлы с расширением .java , а при запуске – файлы с расширением .class. В систему тестирования следует отправлять один файл SamplePlayerJava.java

Python: Сборка не требуется.
В текущей директории при запуске должны находиться файлы с расширениями .py и .pyd. В систему тестирования следует отправлять один файл SamplePlayerPython.py.

Кроме того, в архиве материалов есть проектные файлы для некоторых доступных IDE:

Visual C++: Достаточно открыть прилагаемый солюшен, и можно собирать решение и запускать его с отладкой. Солюшен настроен на запуск игрового сервера одновременно с запуском решения. Сервер запускается с параметром `--debug`: вы можете прописывать параметры для запуска игрового сервера в свойствах проекта GameServer в разделе Debugging.

IntelliJ IDEA: Прилагается директория, которую можно открыть в IDEA, чтобы собирать и запускать решение с отладкой. Запускается только решение: игровой сервер нужно запускать самостоятельно.

PyCharm: Прилагается директория, которую можно открыть в PyCharm, чтобы запускать решение в нём. Запускается только решение: игровой сервер нужно запускать самостоятельно.

8 Запуск решений

Есть два способа запускать решения. Первый и основной способ — указать решение как параметр командной строки игровому серверу, чтобы он сам запустил решение:

```
NsuGameServer.exe SamplePlayerCpp.exe
NsuGameServer.exe "java SamplePlayerJava"
NsuGameServer.exe "python SamplePlayerPython.py"
```

Второй способ — это запустить сначала игровой сервер `NsuGameServer.exe`, а вскоре после этого запустить ваше решение обычным образом (или в обратном порядке). При таком независимом запуске игровой сервер и решение свяжутся друг с другом и также начнётся игра. Этот способ удобнее тем, что вы можете запускать ваше решение обычным образом, в том числе из какой-нибудь IDE.

Игровой сервер запускается с визуализатором (см. раздел 12), в котором можно наблюдать за ходом игры. Кроме того, можно просмотреть игру ещё раз по записанной истории, для этого нужно указать имя файла с историей в командную строку:

```
NsuGameServer.exe current.replay
```

При запуске игрового сервера можно указывать дополнительные ключи:

- `--level=N` или `-l=N` — запустить игру не с начала, а с игрового уровня номер N.
- `--debug` или `-d` — запустить в режиме отладки (см. раздел 9).
- `--seed=N` или `-s=N` — указать ключ инициализации для генераторов псевдослучайных чисел (по умолчанию равно нулю).
- `--winsize=N` — изменить размер окна визуализатора (по умолчанию $N = 700$).

Обратите внимание на **возможность запускать игру с заданного места** `--level`: она может существенно сократить время на отладку решения. Игровые уровни нумеруются подряд в порядке их прохождения, начиная с единицы. Номер текущего уровня отображается в визуализаторе сверху справа после слова `Level`. Можно запустить с какого-либо игрового уровня только после того, как вы хоть раз дошли до него без использования ручного управления. Информация о том, до какого уровня вы дошли, хранится в файле `level.token` в рабочей директории игрового сервера: его следует беречь.

9 Отладка

Чтобы отлаживать решение, рекомендуется:

1. Запускать решение напрямую в отладчике, а игровой сервер запускать отдельно (это второй вариант запуска из раздела 8).
2. Передавать ключ `--debug` при запуске игрового сервера.

Если ключ `--debug` не указан, то когда вы остановите решение в отладчике, игра будет продолжаться. Если ключ указан, то любая остановка решения блокирует игровой сервер.

10 Выходные файлы игры

После запуска игры в рабочей директории создаются файлы различных типов. Вы можете скачать все эти файлы для игр, проведённых на сервере жюри (подробности в разделе 11).

- `gameserver.log` содержит текстовые сообщения, записанные игровым сервером.
- `player.log` содержит текстовые сообщения, записанные программой игрока, в том числе вывод функции `PlayerApi_Log`.
- `current.replay` содержит историю игры, которую можно просмотреть в визуализаторе (см. раздел 12).
- `score.txt` содержит одно целое число: сколько очков набрал игрок.

Текстовые логи полезны для анализа проблем: например, чтобы понять причину внезапной остановки игры. Часть текстовых сообщений дублируется в консоль.

11 Просмотр игр из системы тестирования

Список ваших посылок в систему тестирования NSUts доступен на вкладке «Результаты». Рядом с каждой посылкой есть ссылка «Скачать архив» для скачивания zip-архива. В этом архиве есть текстовые логи, которые можно посмотреть, например, в блокноте. Там же находится файл с историей игры, которую можно проиграть с помощью визуализатора.

12 Визуализатор

С помощью визуализатора можно как запустить новую игру, так и воспроизвести запись уже сыгранной игры. Чтобы воспроизвести запись, нужно передать путь к `.replay`-файлу как параметр командной строки:

```
NsuGameServer.exe current.replay
```

В визуализаторе можно смотреть информацию об игроке и врагах, наводя на них мышку. Можно также кликать на них, чтобы данные выдавались всегда. Информацию о снаряде можно посмотреть только поставив игру на паузу и наведя мышку на снаряд. В информации показаны все поля объекта, которые видит ваше решение. В частности, в первой строке указаны `id` объекта, тип для врага или снаряда. Рядом с игроком выводится сообщение, которое ваше решение записало в поле `message` типа `PlayerControl`, если оно непустое.

Можно ставить игру на паузу, включать/выключать отладочный режим (`--debug`), и включать/выключать ручное управление. Это можно делать в главном меню визуализатора: там же подписаны горячие клавиши.

Ручное управление предназначено для быстрой проверки идей. Оно ограничено 30 секундами игрового времени на один запуск игры. Кроме того, если вы хоть раз переключились на ручное управление, то новые уровни перестают разблокироваться (до перезапуска игры).