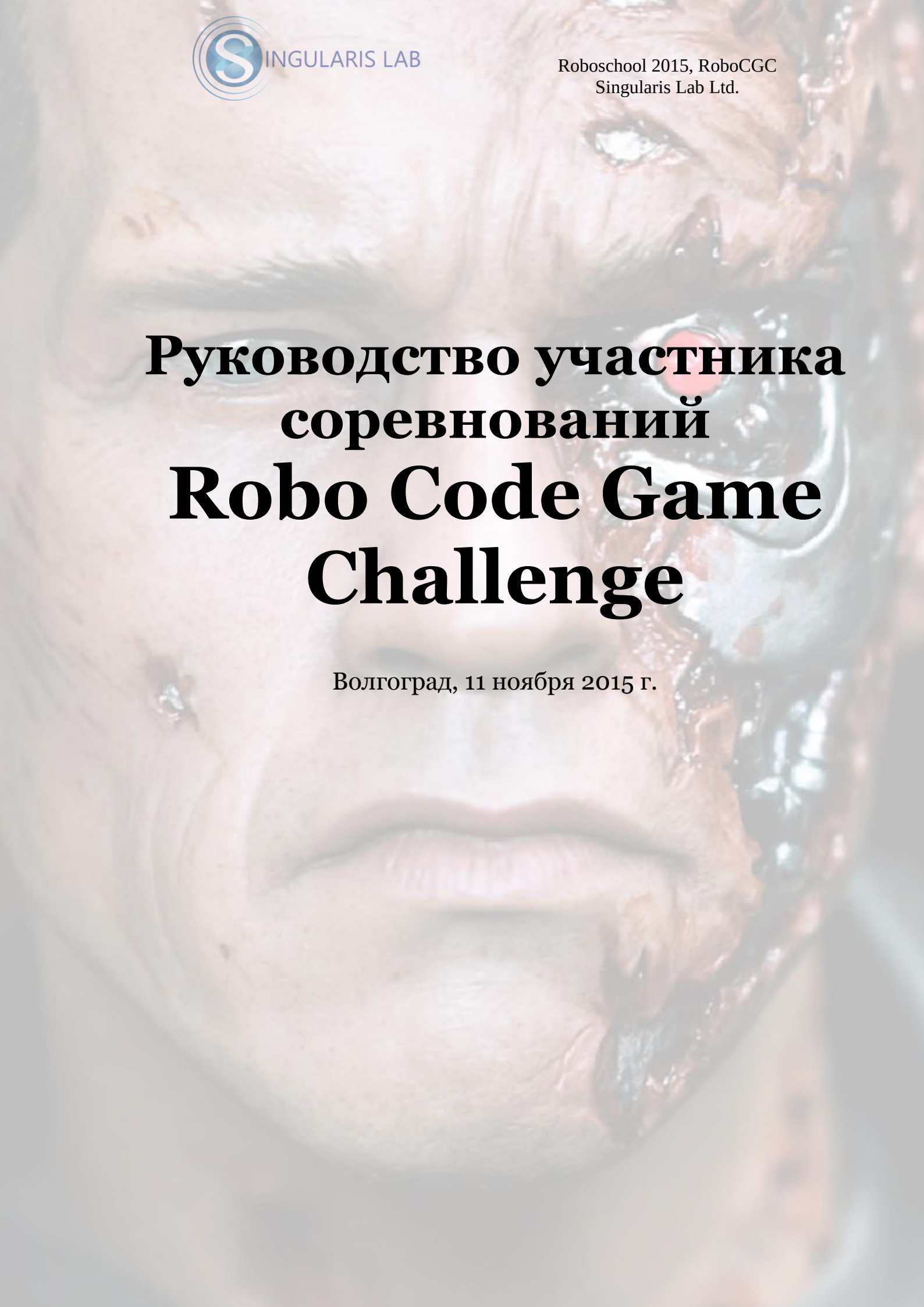

Roboschool 2015, RoboCGC
Singularis Lab Ltd.

Руководство участника соревнований Robo Code Game Challenge

Волгоград, 11 ноября 2015 г.

Предыстория

Развитие технологий к 2xxx году (где xxx > 100) позволило человечеству полностью переложить свои повседневные заботы на плечи роботов: они готовили еду, работали на заводах, убирались на улицах, ухаживали за больными. Единственное, что заботило умы людей – это источник энергии для роботов. И этот вопрос с каждым годом становился все острее и острее.

Но вот был найден ответ. И им оказался новый химический элемент Mozgium (читается как «мозгий»), который добывался в лунных колониях. Но вслед за радостью от открытия пришла новая проблема – многие правители стран решили захватить контроль над его добычей.

Однако, как уже было сказано – сами люди давно ничего не делали, поэтому и воевать отправляли тоже роботов.

Вы – один из «счастливчиков», которым получили реализовать систему управления боевым роботом. В случае если вы преуспеет в этом, то можете рассчитывать на хорошую награду… Ну а если нет – вам лучше поменять место жительства, имя… возможно пол..

Каждая команда должна написать стратегию управления боевым роботом (в вашем распоряжении уникальный ВЕДРОБот-1000 (Вездесущий Едкий Дикий Однофазный Радиоактивный Бот).

Поскольку испытания на реальных роботах является слишком дорогим удовольствием, то на время разработки вам вдается виртуальное окружение для тестирования. Окружение максимально соответствует реальному боту и физике, но могут встречаться отличия.

Боты (ваши и противника) могут передвигаться в плоском двумерном мире. Каждый робот оборудован двумя двигателями, по одному на каждую сторону. Управление каждым двигателем происходит независимо. С помощью подачи различных управляющих воздействий вы можете двигаться вперед (одинаковое положительное управляющие воздействие на оба двигателя), назад (одинаковое отрицательное управляющие воздействие на оба двигателя), поворачивать налево (подача на левый двигатель меньшего управляющего воздействия, чем на правый), поворачивать на право (подача на правый двигатель меньшего управляющего воздействия, чем на правый) и даже развернуться на месте (подача различных по знаку, но равных по модулю управляющих воздействий).

Каждый бот оборудован лазерной пушкой. В начальный момент времени заряд пушки равен 0. Перед выстрелом вам необходимо зарядить ее. После начала заряда вы накапливаете заряд до тех пор, пока не будет

накоплен максимальный заряд. При достижении максимального заряда выстрел происходит автоматически. В любой момент, пока идет заряд вы можете произвести выстрел вручную. После выстрела орудие должно пройти процедуру восстановления и недоступно для накопления нового заряда в течении некоторого времени.

После того как робот произвел выстрел его снаряд летит со скоростью света до первой преграды на своем пути. Если этим препятствием окажется другой робот, то он получает повреждения пропорционально текущему заряду (float(Заряд / Максимальный Заряд * Урон при максимальном заряде)). Робот, который произвел выстрел в данном случае получает очки, пропорционально нанесенному урону.

Каждая команда получает под свое управления по одному роботу.

Кроме роботов в окружении могут присутствовать препятствия. Каждое препятствие представляет собой цилиндрический объект. При столкновении в препятствие, робот может передвигать его.

Правила

Игра проводится следующим образом: каждая команда делает ход. Игровое время дискретно и на каждом временном такте каждый команда делает ход. Во время первого хода игровое время равно `0`. По достижении нулевой отметки времени игроки появляются в игровом мире. Продолжительность игры - до 4 000 тактов времени.

На любое действия игрока дается 0,25 секунды. Если алгоритм не укладывается в это время, то его исполнение будет прервано.

Игровой мир представляет собой прямоугольник, размер которого может отличаться от раунда к раунду. С помощью API игроки могут получить размер игрового мира для текущего раунда.

Размер игрового мира не зависит от разрешения экрана.

Выигрывает игрок, который набрал наибольшее количество очков.

Программирование игрока

Обзор

Вашей команде будут предоставлены заготовки проектов для каждого разрешенного языка программирования. Ваша задача – реализоваться класс `Player`. **Важно: не переименовывайте этот класс!**

В заготовке класса находится 2 метода-заглушки. Вы должны написать реализацию этих методов.

Реализация

После инициализации всех игроков, симулятор начинает вызывать метод Move() для каждого игрока по очереди. Именно программируя этот метод, вы определяете поведение юнитов вашей команды во время игры.

В течении первых 10 тактов вы можете установить имя своей команде с помощью вызова методе SetName(). В случае многократного вызова будет использоваться имя, указанное при последнем вызове. Все вызовы этого метода, совершенные после 10 такта игнорируются.

Основная идея алгоритма – это то, что ваш игрок реально не совершает никакие действия. Он только сигнализирует желание их совершить. После того как симулятор выполнит вызов метода `Move()` для каждого игрока на текущем такте времени, он совершает действия, указанные вашим алгоритмом. Во время этой фазы накладываются все ограничения на совершаемые действия. После того как обрабатываются все действия начинается следующий такт времени.

Симулятор выделяет ограниченное количество времени (0.25 сек) для работы вашего кода. Если ваш код не укладывается в эти рамки, то его выполнение прерывается. Так же выполнение прерывается, если происходит Run-time ошибка или исключение в вашем коде.

2.4 Клиент Code Game Challenge

Вы можете запустить клиент Code Game Challenge, щелкнув по соответствующему ярлыку на рабочем столе вашего компьютера. Используйте выданные вам пароль и логин для авторизации в клиенте.

Клиент позволяет вам тестировать своего игрока и отсылать ваш код на сервер.

Для тестирования вашего кода вы должны использовать закладки Local Test или Global Test клиента.

2.5 Пример кода игрока:

```
void Player::Move(IWorld* world)
{
	// установим имя команды
	if (world->GetTick() == 0)
		world->SetName("Mover");

	if (world->GetOwnUnits().size() == 0)
		return;

	//получим наш юнит
	IControllableUnit* unit = world->GetOwnUnits()[0];

	//каждый 250 тактов будем менять направление движения
	if (world->GetTick() % 500 < 250)
	{
		unit->SetLeftControl(1);
		unit->SetRightControl(1);
	}
	else
	{
		unit->SetLeftControl(-1);
		unit->SetRightControl(-1);
	}

	// если орудие готово к заряду - заряжаем его
	if (unit->ChargingStatus() == 0)
		unit->StartCharging();

	// если орудие зарядилось до уровня 10 - стреляем
	if (unit->ChargingStatus() == 10)
		unit->Fire();
}
```

Интерфейсы программирования

Классы

Class IPlayer

Абстрактный класс игрока. Участникам необходимо реализовать класс-наследник, определяющего метод `Move()`

Методы:

- **`void Move(IWorld* world)`**
  Абстрактный метод для совершения хода. Вызывается на каждом такте симуляции. Этот метод должен быть переопределен в классе-наследнике. Имеет ограничение на время исполнения

Интерфейсы

Интерфейс IWorld

Этот интерфейс поставляет информацию об игровом мире. Интерфейс содержит методы для получения информации о своих юнитах, юнитов противников, размере мира и времени, прошедшем с начала раунда.

Методы:

- **`std::vector<IUnit*> GetEnemies()`**
  Возвращает массив юнитов врагов. Враги располагаются в массиве в случайном порядке. Используя методы классов `IUnit` и `IObject` можно получить подробную информацию о каждом враге.

- **`std::vector<IControllableUnit*> GetOwnUnits()`**
  Возвращает массив собственных юнитов. Юниты располагаются в массиве в случайном порядке. Используя методы классов `IObject` и `IUnit` можно получить подробную информацию о каждом юните. С помощью методов класса `IControllableUnit` можно управлять ими.

- **`std::vector<IObject*> GetObstacles()`**
  Возвращает массив препятствий. Препятствия располагаются в массиве в случайном порядке. Используя методы класса `IObject` можно получить подробную информацию о каждом препятствии.

- **`double GetWidth()`**
  Возвращает ширину игрового поля. Это значение константно и не зависит от выбранного разрешения экрана

- **`double GetHeight()`**
  Возвращает высоту игрового поля. Это значение константно и не зависит от выбранного разрешения экрана

- **`unsigned int GetTick()`**
  Возвращает количество тактов, прошедших с начала игры. В момент первого хода (первый вызов `Player::Move`) равен 0.

- **`void SetName(const std::string name)`**
  Этот метод служит для установки названия команды. Задание имени возможно только в течении первых 10 тактов. Имя должно содержать от 2 до 29 символов.

- **`int GetCurrentTeam()`**
  Возвращает идентификатор команды игрока

Интерфейс **`IObject`**

Этот интерфейс описывает некоторый объект в игровом мире. Объект характеризуется положением, направлением, скоростью, радиусом. В этом интерфейсе находятся вспомогательные методы для вычисления расстояний и углов между объектами.

Методы:

- **`double GetX()`**
  Возвращает координату объекта по оси X.

- **`double GetY()`**
  Возвращает координату объекта по оси Y.

- **`double GetAngle()`**
  Возвращает угол поворота объекта. Предполагается, что у объекта есть некоторое фронтальное направление.

- **`double GetRadius()`**
  Возвращает радиус объекта.

- **`double DistanceTo(IObject *to)`**
  Возвращает расстояние от центра объекта до указанного объекта.

- **`double DistanceTo(double x, double y)`**
  Возвращает расстояние от центра объекта до указанной точки.

- **`double AngleTo(IObject *to)`**
  Возвращает относительный угол в радианах, до указанного объекта. Угол отсчитывается относительно текущего фронтального направления объекта.

- **`double AngleTo(double x, double y)`**
  Возвращает относительный угол в радианах, до указанной точки. Угол отсчитывается относительно текущего фронтального направления объекта.

Интерфейс **IUnit**

Этот интерфейс описывает юнит. Он содержит методы для получения запаса запас прочности юнитов, идентификатора команды и прочих параметров.

Методы:

- **`int GetHP()`**
  Возвращает текущий запас прочности юнита.

- **`int GetMaxHP()`**
  Возвращает максимальный запас прочности юнита.

- **`int ChargingStatus ()`**
  Возвращает статус заряда орудия. Доступны следующие значения:

- < 0 – оружие восстанавливается после выстрела. Модуль значения соответствует остатку времени до возможности использовать орудие.
- 0 – орудие не заряжается;
- > 0 – идет зарядка орудия.

- **`unsigned int GetTeam()`**
  Возвращает идентификатор команды, к которой принадлежит юнит.

Интерфейс `IControllableUnit`

Этот интерфейс содержит методы для управления юнитом.

Методы:

- **`void StartCharging()`**
  Устанавливает флаг начала заряда орудия. Если орудие уже заряжается или находится в состоянии восстановления данный вызов игнорируется.

- **`void Fire()`**
  Устанавливает флаг выстрела из орудия.

- **`void SetLeftControl(double control)`**
  Устанавливает желаемую скорость для левого двигателя. Допускаются значения от в диапазоне [-1; 1], где отрицательные значения соответствуют движению назад.

- **`void SetRightControl(double control)`**
  Устанавливает желаемую скорость для правого двигателя. Допускаются значения от в диапазоне [-1; 1], где отрицательные значения соответствуют движению назад.

Константы

- **Диаметр робота – 0.15м**
- **Максимальный заряд (в игровых тиках) – 20**
- **Время восстановления орудия (в игровых тиках)– 75**
- **Урон при максимальном заряде - 10**