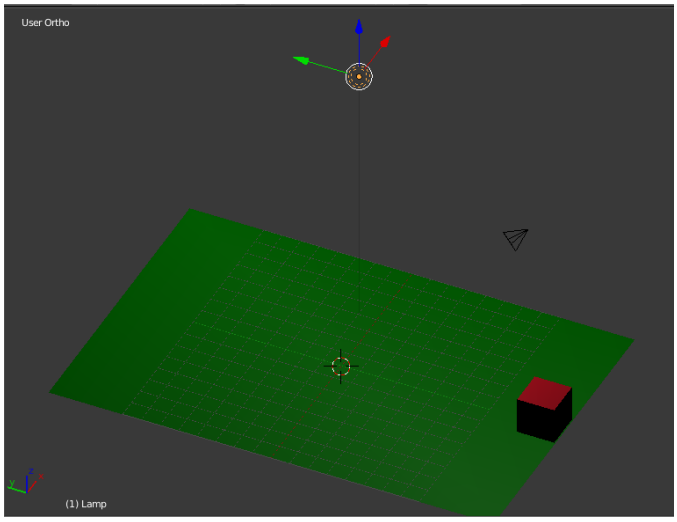
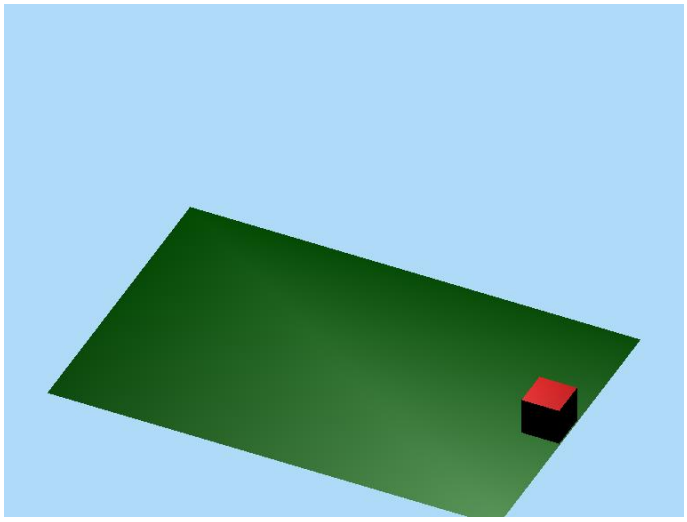
Действительно, Navigation Mesh (дословно – навигационная сетка) является очень удобным инструментом в простых игровых проектах. Ведь, если вам не нужны какие-то слишком умные действия от «врагов» или «соперников», проще заставить их передвигаться по заранее намеченным путям, чем писать сложную логику на Python. Здесь мы рассмотрим несколько примеров применения навигационной сетки, которая позволяет объекту не только перемещаться к цели по определённому пути или кратчайшему расстоянию, но и обходить препятствия на пути следования.

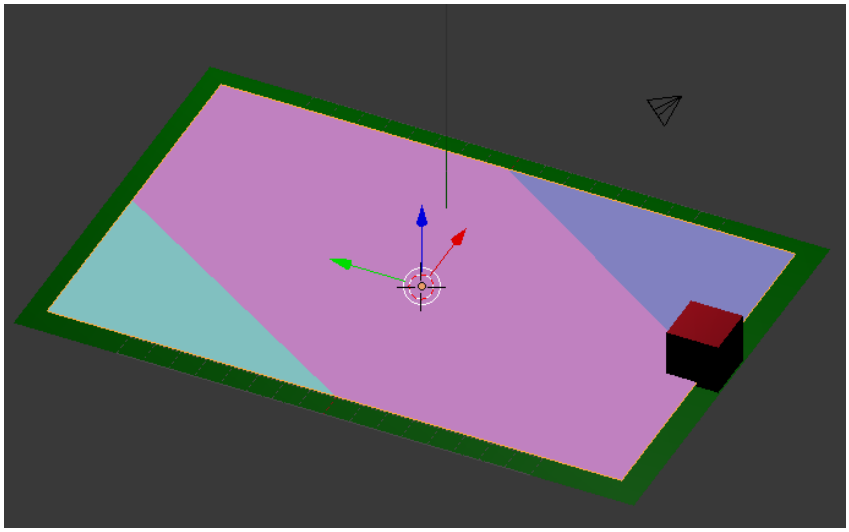
Оставим настройки сетки по умолчанию. В будущем, возможно, вам захочется настроить её вручную. А сейчас давайте оставим всё, как есть. Учтём лишь то, что ширина сетки по умолчанию должна быть больше стандартного куба, т.е. единицы (1.000).

Начнём с простого – создадим плоскость с размером X-8.000 Y-12.000 Z-1.000. Куб сместим к одному краю плана и приподнимем над ним. Визуально он должен «лежать» на плане. Раскрасим план в зелёный цвет, куб в красный цвет, небо(окружение) – в голубой. Не забываем перейти в игровой режим (Blender Game). Наконец, приподнимем свет над планом на 11,500 единиц (все размеры указаны в blend-единицах):





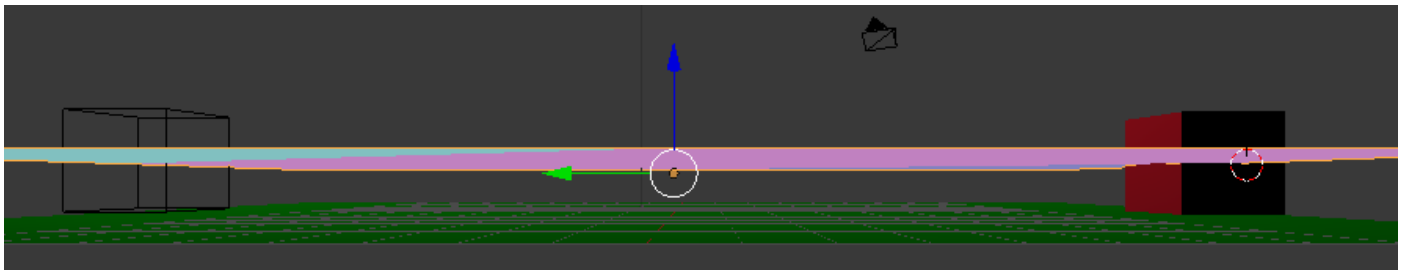
Теперь выделим план и перейдём на вкладку Scene, где, в разделе Navigation mesh, нажмём большую кнопку Build navigation mesh (создать навигационную сетку). Blender создаст сетку автоматический, что выразится в дополнительной разноцветной плоскости над основной:



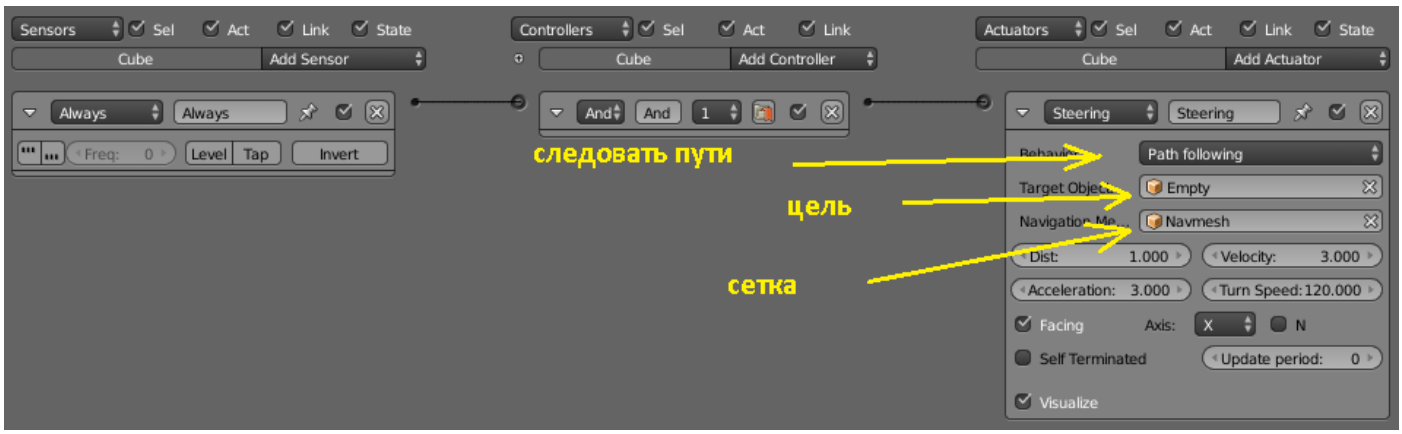
Это и есть наша навигационная сетка. В процессе игры она не видна и на физику ни как не влияет. Все предметы проходят сквозь неё без сопротивления. Чтобы заставить наш куб двигаться, ему нужно задать направление. Иначе, куда он будет двигаться? Для этого просто необходимо установить ещё один объект – цель. Целью может быть любой объект. Если нам нужно отслеживать столкновения с целью, то это должен

быть физический объект. Если столкновения не важны, то можно применить обычную пустышку Empty. Для нашей цели пока вполне достаточно пустышки куба. Установим её на противоположном краю плана:

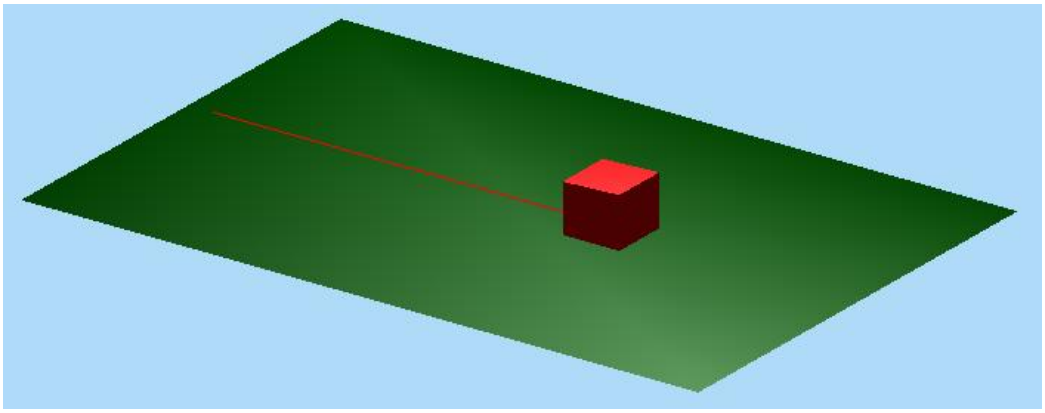
Следует учитывать, что центр игрового объекта будет стремиться к плоскости навигационной сетки. Поэтому, если объект игрока у вас типа Static, то он к концу пути может начать погружаться в основную плоскость. Чтобы этого не произошло, есть смысл выделить навигационную сетку и приподнять её до середины игрового объекта:



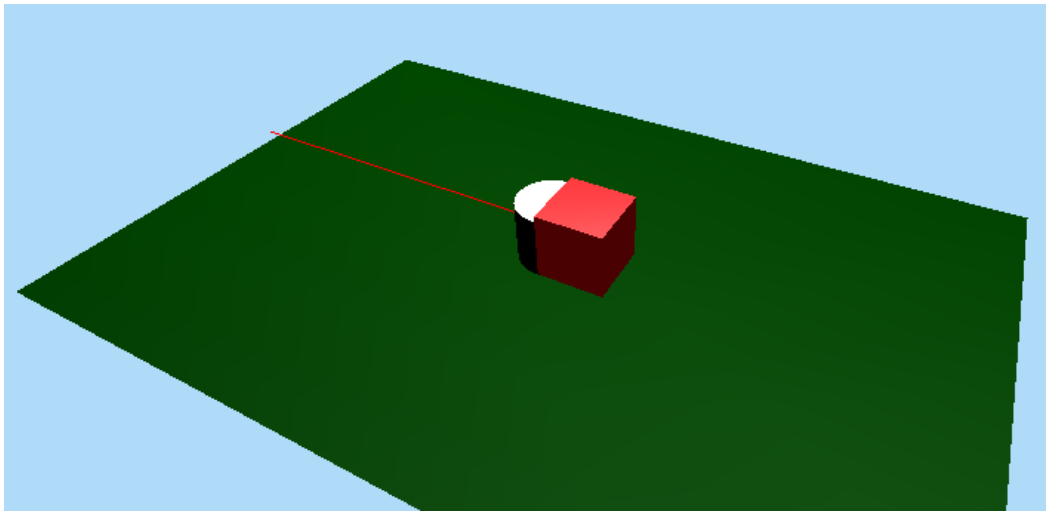
При работе игры пустышка не видна. Это тоже следует учитывать. Теперь перейдём в игровую логику. Выделим куб-игрока. Вставим сенсор **Always**, контролёр **And** и актуатор **Steering**, в котором выставим свойство Behavior как Path following. В качестве цели назначим пустышку, а в качестве сетки выберем нашу созданную сетку . Все имена объектов вы можете изменить. Я же оставил по умолчанию:



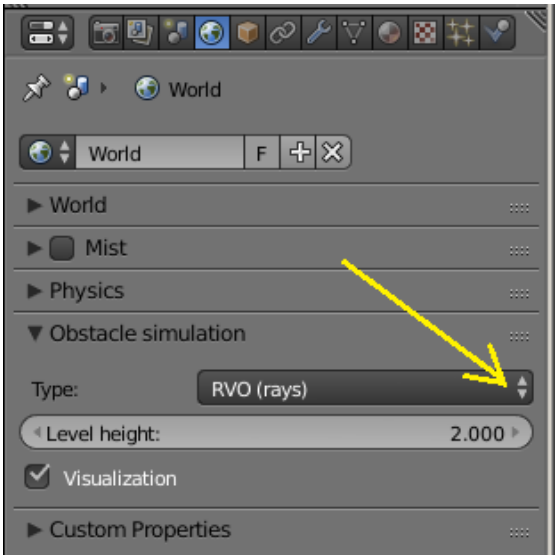
Dist – дистанция от центра объекта до центра цели после того, как цель достигнута. Velocity – скорость перемещения объекта. Для статического (Static) объекта этих параметров достаточно. Axis назначает фронтальную ось, проще говоря, каким боком вперёд будет перемещаться объект. Запускаем проект лат. «P» и наблюдаем картину перемещения:



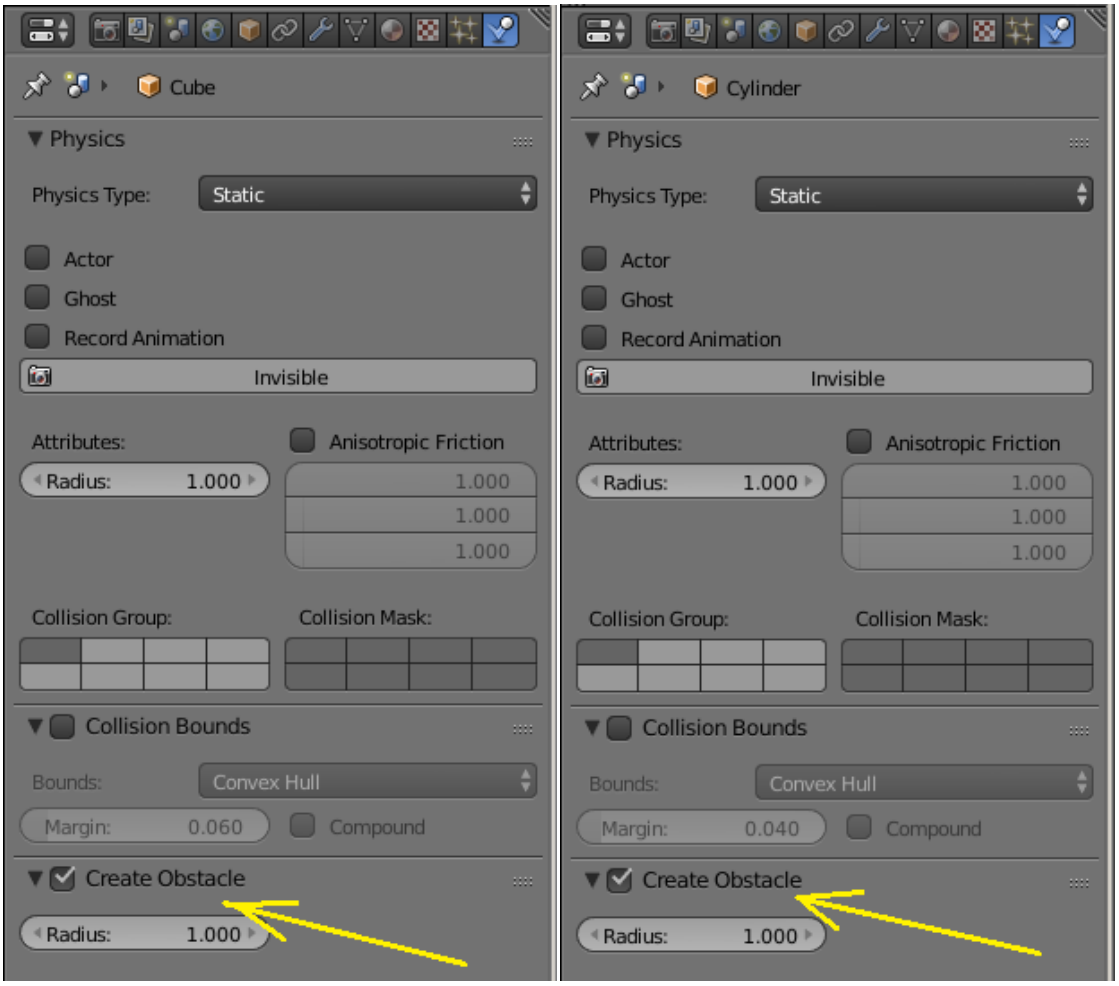
Я включил отображение направления пути для наглядности. Направление отмечено красной линией. Давайте поставим на середине пути ещё один объект – цилиндр. Запустим проект и убедимся, что куб пройдёт сквозь цилиндр:



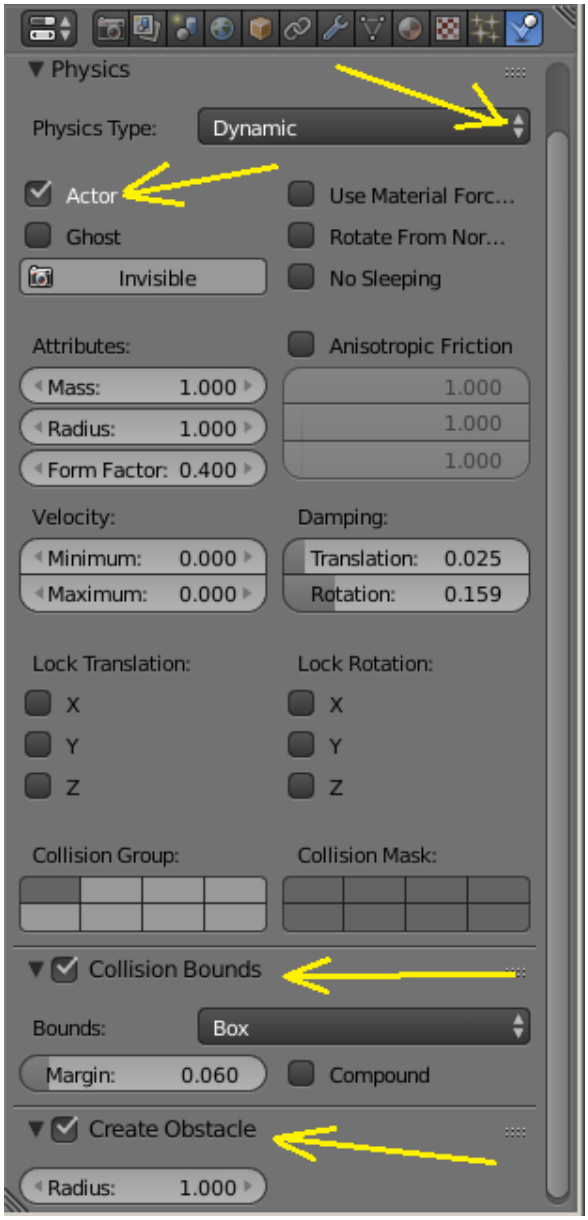
Как заставить куб огибать препятствие ? Для этого у Blender есть одноимённая функция Obstacle (препятствие), которую необходимо включить. Её основа находится на вкладке World:



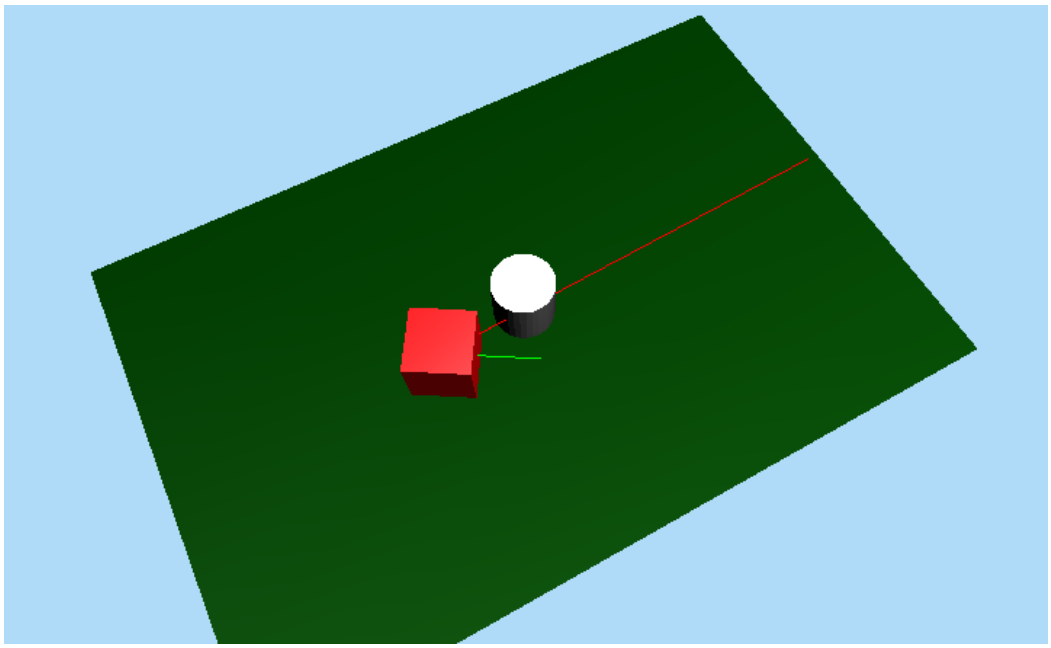
Тоже самое присутствует на вкладке Physics каждого объекта:



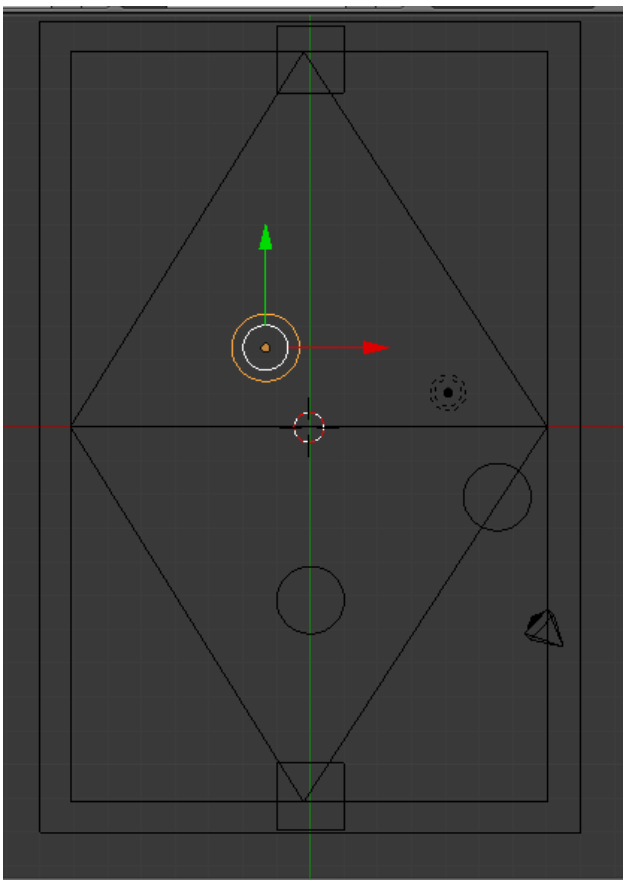
Давайте включим Obstacle у куба и цилиндра, и посмотрим, что из этого выйдет? Куб действительно пытается обойти препятствие, но делает это ужасно и некорректно. В чём дело? Можно, конечно, увеличить радиус обхода препятствия у цилиндра с 1 до 3. Но это только усугубит ситуацию. Дело в том, что достаточно корректно могут обсчитываться только физические объекты, с включённой функцией обнаружения столкновений. При этом движущийся объект должен быть если не Rigid Body , то хотя бы Dynamic (динамичный, движущийся…). Давайте включим столкновения у куба (смотри ниже):

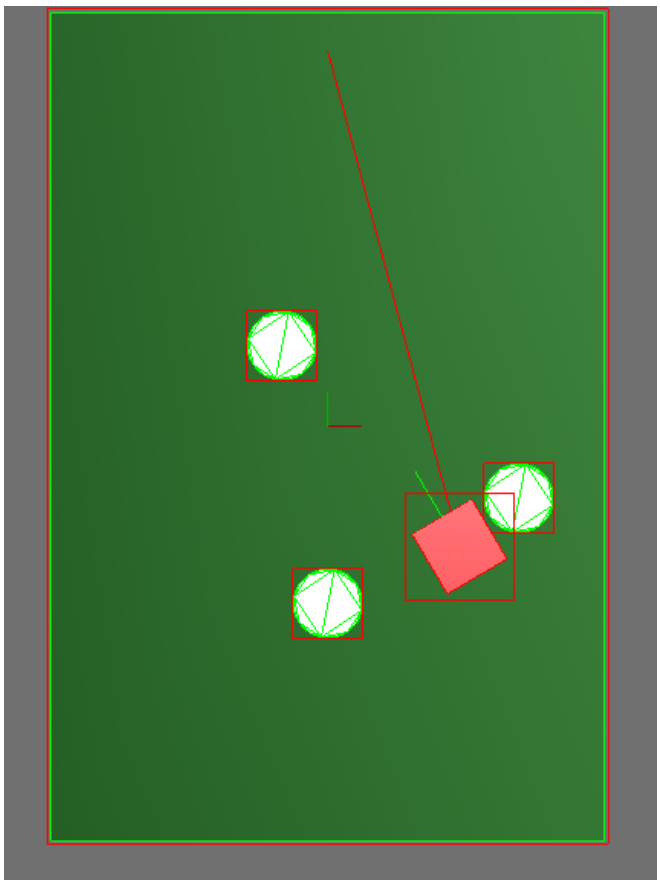


А теперь запустим проект и убедимся, что куб аккуратно огибает цилиндр. Если у куба немного увеличить радиус Obstacle, то куб постарается даже не касаться цилиндра:

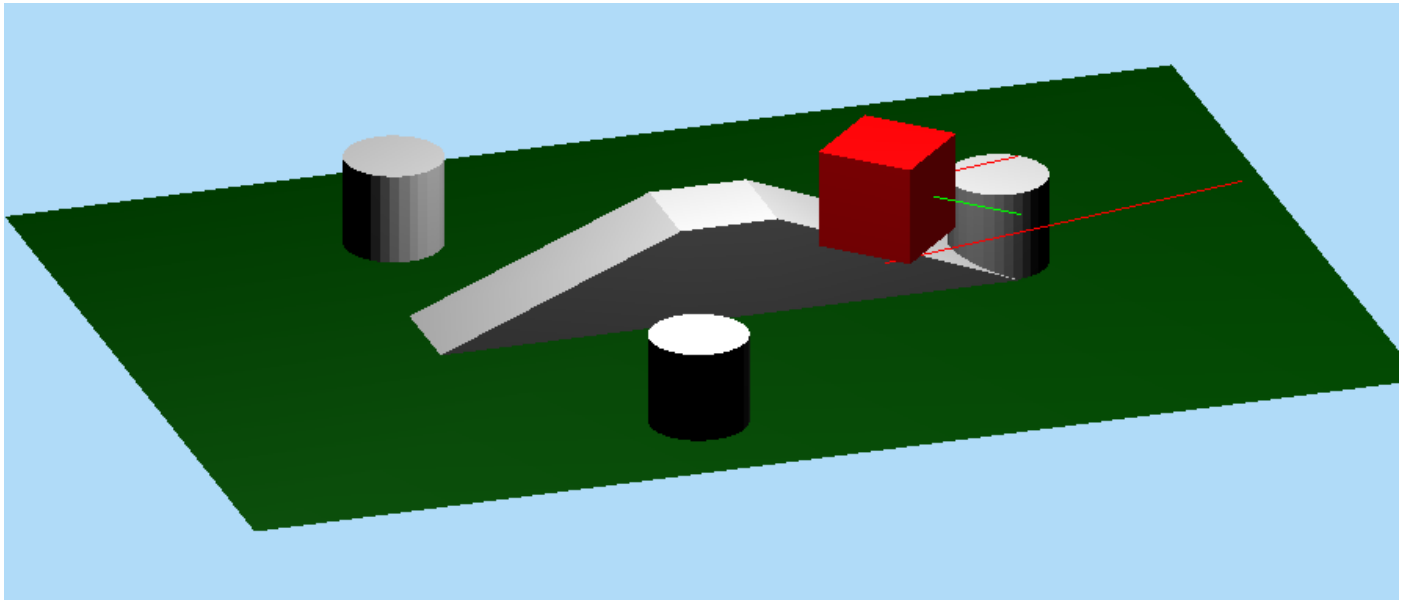


Давайте сдвинем цилиндр ближе к кубу и проставим Obstacle у куба = 1.600, а у цилиндра =1.500. Продублируем цилиндр два раза и разместим его копии чуть дальше на пути куба. Запустим проект:





Как видим, куб справляется с задачей и встретив препятствие обходит его. Поставим ещё одно препятствие в виде дополнительного элемента ландшафта – горки. Obstacle у горки включать не нужно. Проверим в действии:



Встретив на пути горку, наш куб спокойно по ней взбирается и в конце обходит цилиндр, направляясь к цели.

Здесь мы лишь рассмотрели сам принцип применения навигационной сетки. Подключив чуточку фантазии и смекалки ей можно найти разнообразное применение. Ведь сетку можно построить и на более сложном объекте: горке, лестнице, дорожке, тоннеле и т.д.

Урок составил Niburiec для сайта http://blender-game.ucoz.ru/ 20.01.2015.