Работа #1 - Компонент физики Rigidbody в Unity

Привалов Вадим РИ-230914.

цель работы: изучить свойства компонента Rigidbody, освоить реализацию игровых механик с применением свойств этого компонента



## Задания к работе

### Задание 1

1.1 Создайте сцену, изображенную на рисунке ниже:

| * CubeGround - земля * Stage - блок ступени на земле * CubeSid - тёмный куб размером 1х1 метр * CubeNancy - светлый куб размером 1х1 метр |  |
| --- | --- |

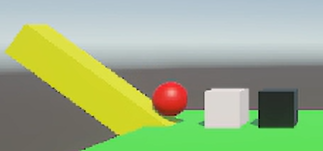
* 2.2 Добавьте компонент Rigidbody на объекты CubeSid и CubeNancy. Запустите сцену, проверьте что кубики падают вниз и останавливаются на земле.

2.3 Установите массу для объекта CubeSid = 1 кг, для объекта CubeNancy = 100 кг. Запустите сцену. Какой из двух кубов быстрее достигнет земли и почему? Дайте развернутый ответ:

| После установки массы в 100 кг объекту CubeNancy и запуску программы я увидел что скорость полета у объекта CubeNancy быстрее чем у CubeSid так как объект CubeNancy тяжелее, ну и объекту CubeSid мешает другой объект Stage (лично у меня так стоит чтобы другой кубик задевал платформу). |
| --- |

* 2.5 Измените значение свойства Drag внутри компонента Rigidbody так, чтобы CubeSid достигал земли быстрее чем CubeNancy.

2.6 Дополните сцену сферой и эстокадой, по которой будет скатываться сфера с компонентом физии твердого тела Rigidbody. Установите массу сферы равную 100 кг, размер оставьте по умолчанию. Поместите сферу в верхней части эстокады, а кубы Sid & Nancy будут помещаться в нижней части с целью тестирования механики:



2.7 Протестируйте наезд сферы на кубы разной массы (1 и 100 кг). Опишите характер столкновения, влияет ли параметр Drag (если да, — то как?)

| Объект CubeSid при ударе сферы отлетает в сторону так как масса объектов разная.Объект CubeNancy просто останавливается на месте.  Высокий Drag - объект быстро останавливается после столкновения.  Низкий Drag - объект долго скользит. |
| --- |

2.8 Модифицируйте сцену для тестирования параметров Drag и Angular Drag:

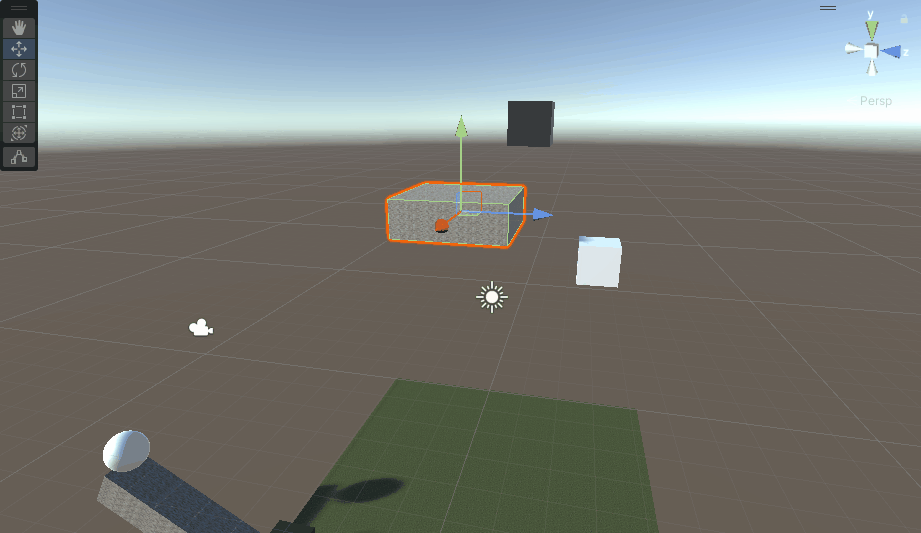
| Drag = 0, Angular Drag = 0.05 | Drag = 2, Angular Drag = 2 |
| --- | --- |
|  |  |

2.9 Протестируйте, что произойдет если указать разные значения для Drag и Angular Drag. Имеет ли это какую-то практическую ценность? Почему по умолчанию значение Angular Drag близко к нулю, но не равно ему? Проверьте что произойдет, если указать значение Angular Drag = 0. Дайте развернутый ответ:

| Случай 1: Drag = 0, Angular Drag = 0.05  Движение: Объект не замедляется (летит или скользит бесконечно, если нет трения).  Вращение: Медленно останавливается из-за небольшого Angular Drag = 0.05.  Пример:  Если толкнуть куб, он поедет без остановки, но его вращение постепенно затухнет.  Случай 2: Drag = 2, Angular Drag = 2  Движение: Объект быстро тормозит (как будто движется в вязкой среде).  Вращение: Останавливается почти мгновенно.  Пример:  Если кинуть такой куб, он пролетит немного и резко остановится, почти не вращаясь.  Случай 3: Angular Drag = 0 (экстремальный случай)  Вращение: Объект вращается бесконечно без замедления.  Пример:  Если раскрутить сферу, она будет крутиться вечно (как в космосе без трения).  На практике это выглядит неестественно, поэтому в Unity Angular Drag по умолчанию 0.05 (очень маленькое, но не нулевое). |
| --- |

* 2.10 Назначьте на платформу Stage компонент Rigidbody и отключите гравитацию (Use Gravity). Проверьте, что произойдет при запуске сцены.

2.11 Во вкладке Project Settings - Physics увеличьте значение силы гравитации. Подберите такое значение силы, при котором скорость падения достаточно большая для того, чтобы движок не успел обработать столкновение кубов Sid & Nancy с платформами Stage и Ground. В некоторых случаях можно добавиться интересного эффекта:



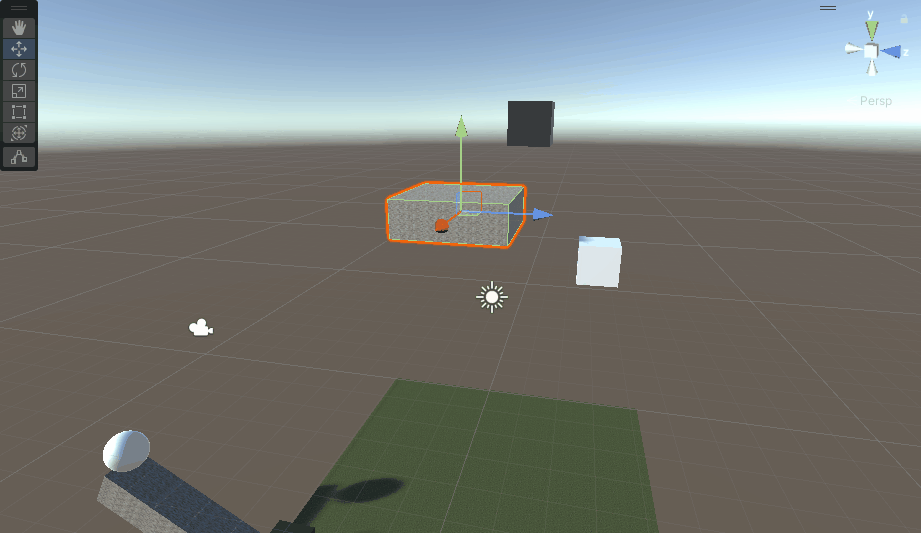
*в примере значение гравитации -90,8, однако, движок успевает обработать столкновения*

Напишите найденное значение и от чего оно зависит (по вашему мнению):

| Мое значение это -150,зависит от **Fixed Timestep** (частота расчёта физики),  Режима детекции столкновений (Discrete/Continuous)/ |
| --- |

* 2.12 Верните значение гравитации к настройкам по умолчанию.

2.13 Для объекта Stage зафиксируйте свойство FreezePosition внутри компонента Rigidbody таким образом, чтобы объект вращался вокруг оси, не падая вниз. Укажите, как должна выглядеть настройка FreezePosition?



| В моем случае я поставил галочки на X,Y,Z и отключил isKinematic у меня все заработало так как показывает картинка выше. |
| --- |

2.14 isKinematic позволяет сделать объект нефизическим. Если поставить галочку isKinematic, — то объект будет жестко стоять на месте. При этом управлять кинематическими свойствами объекта можно во время выполнения сцены. Запустите сцену повторно и активируйте свойство isKinematic на объекте Stage во время вращения. Что произойдет, если галочку вновь отключить при запущенной сцене? Продолжит ли платформа своё вращение?

| Если нужно **временно заморозить** вращение с возможностью возобновления — лучше не использовать isKinematic.При отключении isKinematic вращение не продолжится — потребуется явно задать скорость заново.  Для динамических объектов используйте альтернативные методы (сохранение скорости, отключение сил). |
| --- |

2.15 Поднимете объект SidCube на высоту 1000 метров и запустите сцену. Дождитесь момента, когда куб достигнет земли Ground при падении. По какой причине куб проскакивает землю, не останавливаясь? Какие могут быть способы решения этой проблемы?

| Основная причина:  Слишком высокая скорость падения приводит к тому, что между кадрами физики (FixedUpdate) куб перемещается дальше, чем толщина коллайдера Ground. Движок Unity просто "не успевает" обнаружить столкновение.  Решения:  Увеличить толщину коллайдера Ground  Сделайте коллайдер Ground толще (например, 5–10 м).  Уменьшить Fixed Timestep  Edit → Project Settings → Time → Fixed Timestep (например, уменьшите до 0.005). |
| --- |

2.16 В скрипт файле ниже приведён пример, в котором происходит управление свойствами компонента RigidBody при нажатии на кнопку Space. Для работы созданный скрипт-файл следует подключить к объекту, свойствами которого планируется управлять при нажатии на клавишу Space:

| using UnityEngine;  public class RigidbodyMod : MonoBehaviour {  private Rigidbody \_rb;  void Start()  {  \_rb = GetComponent<Rigidbody>();  }   void Update()  {  if(Input.GetKeyDown(KeyCode.Space))   {  //\_rb.isKinematic = true;  \_rb.mass= 1.0f;  \_rb.drag = 0;  \_rb.angularDrag = 0.05f;  \_rb.useGravity= true;  }  } } |
| --- |

* 2.17 Используя этот сценарий и знания, полученные в работе, доработайте игровую сцену и создайте “уникальный интерактив”. Часть действий на сцене должны запускаться при нажатии клавиш клавиатуры.

2.18 Напишите развернутый вывод по проделанной работе. Как знания свойств компонентов RigidBody помогут в создании интерактивных приложений, игр и симуляторов?

| В ходе выполнения задания мы создали и подключили скрипт RigidbodyMod к объекту в Unity, который позволяет управлять свойствами компонента Rigidbody при нажатии на клавишу Space. Этот процесс включал в себя несколько ключевых этапов:  1. Создание и написание скрипта: Мы разработали C# скрипт, который использует компонент Rigidbody, чтобы изменять его свойства в ответ на ввод пользователя. Это позволило нам понять, как взаимодействовать с компонентами Unity через код.  2. Подключение компонента Rigidbody: Мы добавили компонент Rigidbody к игровому объекту, что дало возможность использовать физику в игре. Это важный шаг, так как физические взаимодействия являются основой многих игровых механик.  3. Настройка поведения объекта: Изменяя свойства Rigidbody, такие как масса, drag (сопротивление) и использование гравитации, мы получили возможность контролировать физическое поведение объекта в игре. Это демонстрирует, как простые изменения могут значительно повлиять на взаимодействие объекта с окружающей средой.  Понимание свойств компонента Rigidbody является важным аспектом разработки интерактивных приложений, игр и симуляторов по нескольким причинам:  1. Физическая реалистичность: Компонент Rigidbody отвечает за реализацию физических законов, таких как гравитация, столкновения и движение. Знание того, как правильно настраивать эти свойства, позволяет разработчикам создавать более реалистичные и увлекательные игровые миры. Например, в спортивных играх или симуляторах транспортных средств важно точно моделировать физику для достижения достоверности.  2. Управление взаимодействиями: В играх часто требуется управление взаимодействиями между объектами (например, столкновения). Знание о том, как работают свойства Rigidbody, помогает разработчикам настроить эти взаимодействия, чтобы они были интуитивно понятны и соответствовали ожиданиям игроков. Это может включать настройку силы столкновения, реакции объектов на удар и т.д.  3. Создание уникальных игровых механик: Используя Rigidbody, разработчики могут создавать уникальные механики игры. Например, можно реализовать механики полета, прыжков или катания, изменяя параметры Rigidbody в зависимости от ситуации. Это открывает широкие возможности для креативности и экспериментов.  4. Оптимизация производительности: Правильное использование свойств Rigidbody может также помочь в оптимизации производительности игры. Например, использование isKinematic позволяет отключить физическое взаимодействие для объектов, которые не требуют его, что снижает нагрузку на процессор.  5. Интерактивность и пользовательский опыт: В интерактивных приложениях знание о физических свойствах объектов помогает создавать более привлекательные и отзывчивые интерфейсы. Например, элементы интерфейса могут реагировать на действия пользователя с использованием физики, что делает взаимодействие более увлекательным. |
| --- |