**“Использование новейших технологий программирования компьютерной графики для создания трехмерного продукта для изучение разного вида химических реакций”**

Работу создал:

Мусаев Илькин

Ученик 10 класса

Лицея им. Зарифы Алиевой

Баку-2014

Содержание

Оглавление

[Вступление 3](#_Toc401846868)

[Создание программного продукта Chemistry Reactions 4](#_Toc401846869)

[Видеопроигрыватель 14](#_Toc401846870)

[Теория 16](#_Toc401846871)

[Выводы 17](#_Toc401846872)

# Вступление

Химические реакции являются самым интересным разделом химии. Они позволяют изучать поведение различных элементов в одной среде с получением новых веществ. Но для того что бы увидеть химическую реакцию на молекулярном уровне требуется огромные затраты на дорогое оборудование, а также большой период времени. Исходя из этого, я создал программу Chemistry Reactions.(Рис 1.)

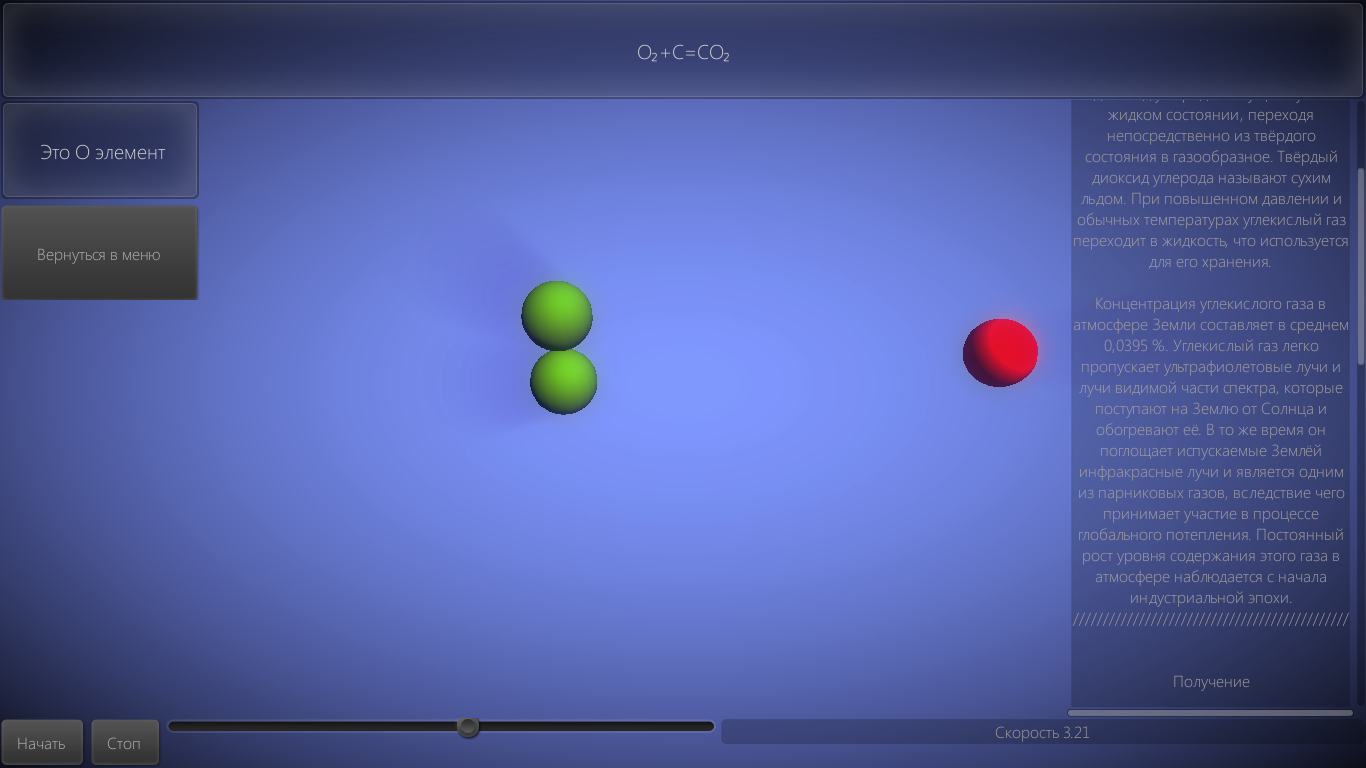


Рис 1. Режим визуализации

Благодаря данной программе вы сможете понять каким образом проходят различные виды химических реакций, получить основную информацию о полученных веществах, а также посмотреть интереснейшие видео о различных химических реакциях.

# Создание программного продукта Chemistry Reactions

Для создания своей работы я использовал Unity3D, эта программа является лидеров по выпуску программного обеспечения для создания игр и программных продуктов. Unity3D может сразу запустить игру и увидеть результат, что позволяет увеличить эффективность работы в разы. (Рис 2.)

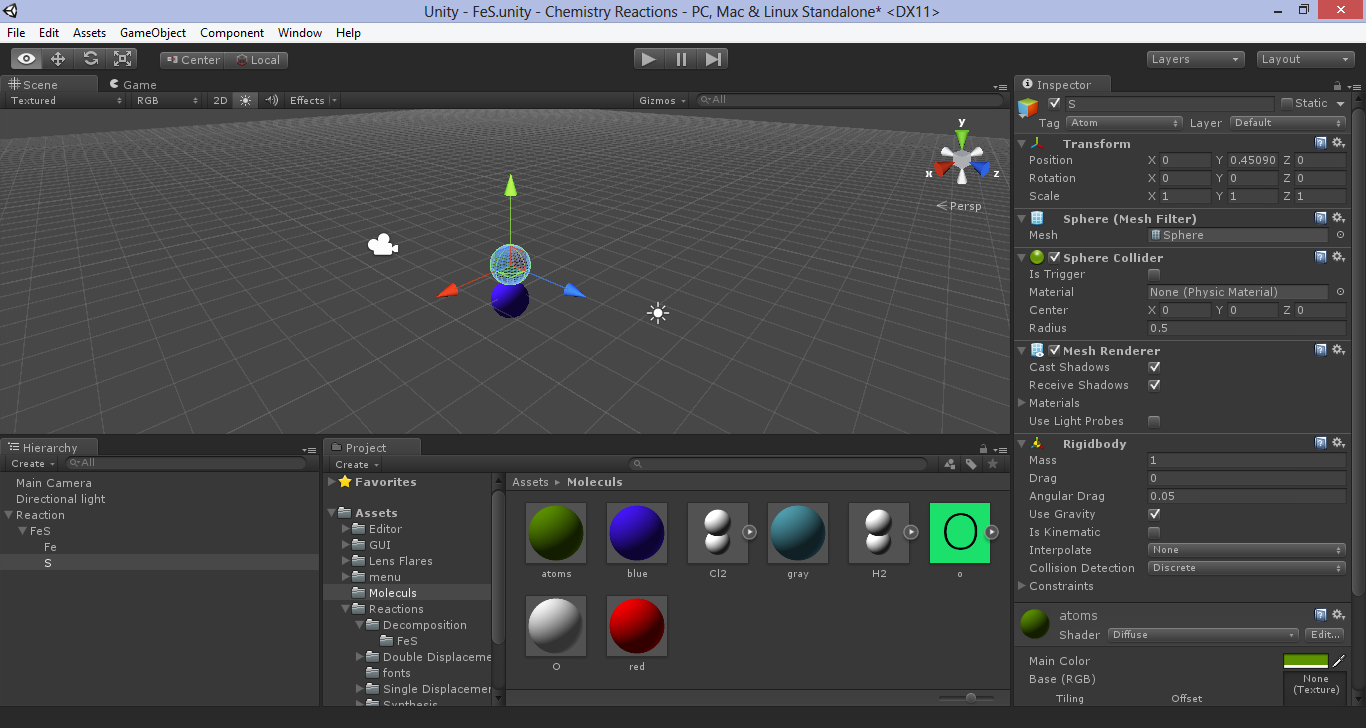


Рис 2. Интерфейс Unity3D

Так как, Unity3D построен на компонентах DirectX, то вы можете считывать данные с любого подключенного устройства. В Unity3D возможно написание на таких языках программирования, как:

* C#
* JavaScript
* Boo

Для написания проекта я выбрал c#, так как он является самым эффективным и удобным языком.

Для реализации системы меню я использовал встроенную в Unity систему – GUI. Она позволяет писать графический интерфейс с помощью кода, что является весьма гибким решением. Пример части кода с использованием GUI:

using UnityEngine;

using System.Collections;

public class Picker\_Controller : MonoBehaviour

{

public GUISkin style;

string current\_element = "Нажмите на любой обьект";

void OnGUI()

{

GUI.skin = style;

GUILayout.BeginArea(new Rect(0, 100, 200, 200));

GUILayout.BeginVertical();

GUILayout.Box(current\_element == "Нажмите на любой обьект" ? current\_element : "Это " + current\_element + " элемент", GUILayout.Height(100), GUILayout.Width(200));

if (Input.GetMouseButtonDown(0))

{

Ray ray = camera.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

if (Physics.Raycast(ray, out hit))

{

if (hit.collider.tag == "Atom")

{

current\_element = hit.collider.gameObject.GetComponent<Atom>().name;

}

}

}

GUILayout.EndVertical();

GUILayout.EndArea();

}

}

C#

Данный код реализует вывод на экран текущего элемента. Вначале мы создаем экземпляр класса GUISkin, который поможет нам определить свои стили для пользовательского интерфейса. На следующей строке мы создаем строку, куда будем записывать название текущего элемента.

После этого мы пишем функцию OnGUI – это главная функция для управления и создания пользовательского интерфейса. Данная функция вызывается при каждом обновлении экрана, что зависит от характеристик каждого компьютера. В среднем это число составляет 20-25 кадров за секунду.

Далее мы указываем, что мы будем определять свои стили. После этого мы создаем контейнер для размещения остальных элементов, указывая ему в качестве параметра координаты левого верхнего угла, а также размеры. Внутри данного контейнера мы создаем горизонтальный блок, внутри которого будет находиться объект Box. В качестве параметров мы указываем ему строку для вывода, а также как дополнительный параметры ширину и высоту (если мы их не укажем блок автоматически подстроится под размер текста).

После этого мы проверяем была ли нажата левая кнопка мыши, если да, то получаем ее координаты. Далее проверяем произошло ли нажатие на объект с тегом “Atom”, если да, то получаем название данного элемента из скрипта Atom, который находится на всех атомах в сцене. Название элемента хранится в публичной переменной, доступ к которой можно совершить из любого c# скрипта.

Код на скрипт Atom:

using UnityEngine;

using System.Collections;

public class Atom : MonoBehaviour {

public string name;

}

C#

Скрипт Atom имеет лишь одну переменную для хранения имени вещества. Данный скрипт добавляется ко всем атомам в сцене данный скрипт в Unity выглядит, так:

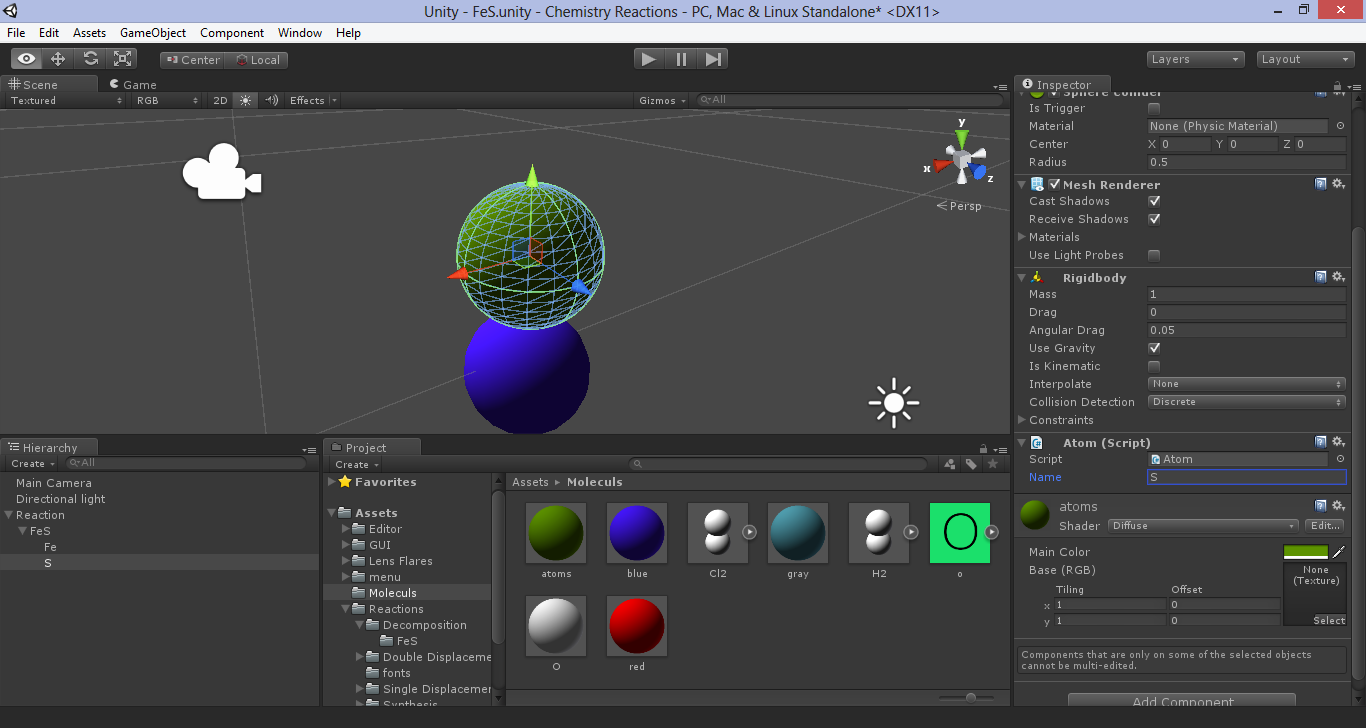


Рис 3. Скрипт Atom в Unity3D

Для расчета взаимодействия между атомами используется движок NVIDIA PhysX, который предлагает более качественный игровой процесс и более яркие впечатления от игры, обеспечивая более реалистичное взаимодействие окружений и персонажей, чем когда-либо. Благодаря более реалистичному поведению графика выглядит и воспринимается намного лучше. Данный движок встроен прямо в Unity3D в виде легко добавляемых компонентов.

На каждый атом добавляется один из компонентов PhysX – Rigidbody, который используется для расчета столкновения и взаимодействия твердых тел.

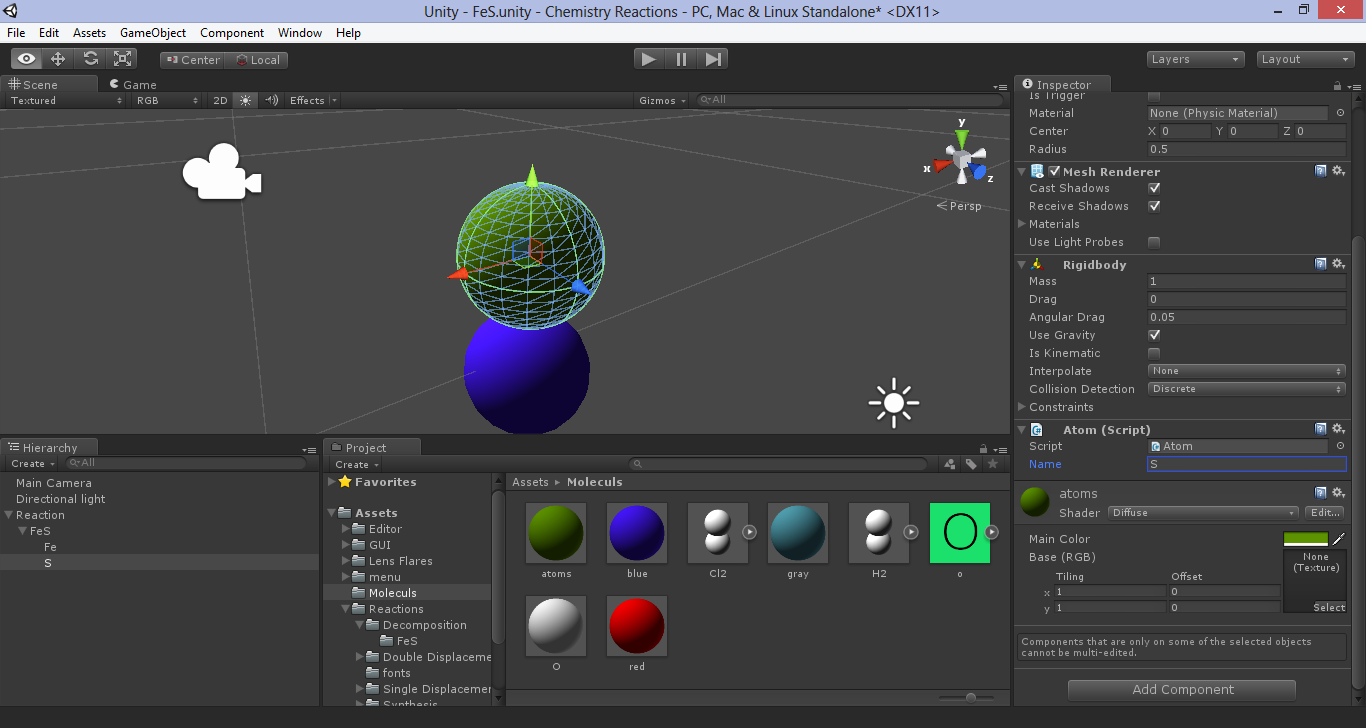


Рис 4. Компонент rigidbody в Unity3D

В данной таблицы находится описание основных свойств данного компонента.

| **Свойство:** | **Функция:** |
| --- | --- |
| Mass | Масса объекта (в произвольных единицах). Вы не должны делать массу больше или меньше 100 раз, чем в других твёрдых телах. |
| Drag | Какое воздушное сопротивление оказывается на объект пока он перемещается под воздействием этих сил. 0 означает отсутствие сопротивления, а бесконечность (infinity) тут же прекращает перемещение объекта. |
| Angular Drag | Какое воздушное сопротивление оказывается на объект пока он вращается под воздействием силы вращения. 0 означает отсутствие сопротивления. Учтите что вы не можете остановить вращение объекта путём установки его углового сопротивления (Angular Drag) в бесконечное (infinity) положение. |
| Use Gravity | При включении на объект действует гравитация. |
| Is Kinematic | При включении, объект не будет управляться физическим движком, и сможет управляться только при помощи своей трансформации. Полезно при перемещении платформ или если вам необходимо анимировать твёрдое тело, которое имеет назначенный HingeJoint. |
| Interpolate | Попробуйте одну из опций если вы замечаете тряску в перемещении своего твёрдого тела. |
| None | Не применено никакой интерполяции. |
| Interpolate | Сглаживание транформации основано на трансформации из предыдущего кадра. |
| Extrapolate | Сглаживание трансформации основано на приблизительной трансформации следующего кадра. |

Для создания связи между атомами используется компонент Fixed Joint.

Компонент Fixed Joints ограничивает движение определенного объекта, связывая его с другим объектом. Чаще всего этот компонент используется в случае, если в определенный момент времени может потребоваться разъединить два объекта, или наоборот, соединить два объекта без необходимости изменения иерархии.(Рис 5.)

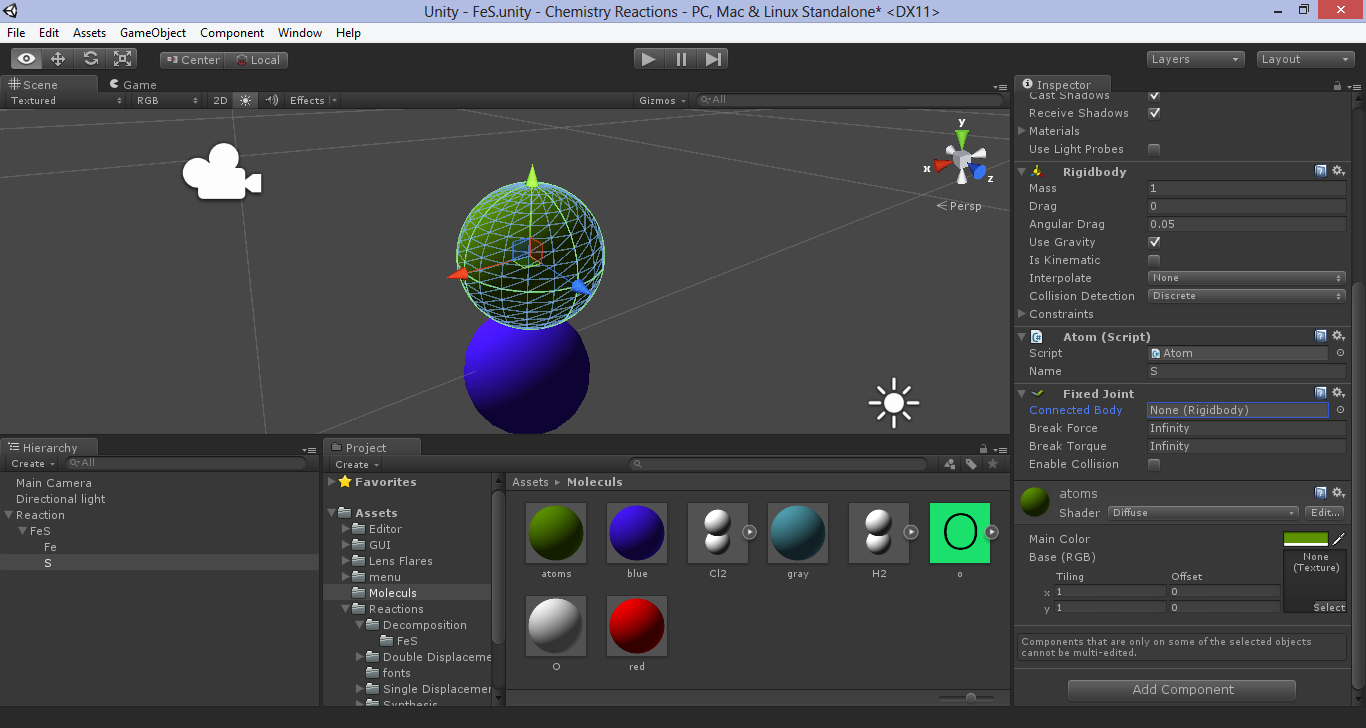


Рис 5. Компонент Fixed Joint в Unity3D

Описание свойств компонента Fixed Joint приведены в следующей таблице:

| ***Свойство:*** | Функция: |
| --- | --- |
| Connected Body | Необязательная ссылка на другой объект с Rigidbody, к которому присоединяется текущий объект. Если поле оставить пустым, объект присоединяется к заданной точке в пространстве. |
| Break Force | Сила, которую требуется приложить к объекту, чтобы разорвать соединение. |
| Break Torque | Крутящий момент, который необходимо приложить к объекту, чтобы разорвать соединение. |
| Enable Collision | Если включено, два соединенных тела будут сталкиваться друг с другом. |

При создании игр иногда возникают случаи, когда требуется, чтобы объекты двигались вместе (временно или постоянно). Компоненты Fixed Joint позволяют упростить реализацию подобных ситуаций, поскольку вам не приходится менять положение объекта в иерархии с помощью скриптов. Минус подобного решения в том, что вам придется добавлять компоненты Rigidbodies на объекты, которые требуется соединить с помощью Fixed Joint.

Для определения пределов прочности соединений можно использовать свойства Break Force и Break Torque. Если их значение меньше чем Infinity (бесконечность) и приложенная к соединению сила оказывается больше этих значений, соединение Fixed Joint разрывается и перестает удерживать два объекта вместе.

В зависимости от типа реакции пишется скрипт на данную реакцию. Для примера рассмотрим скрипт на реакцию FeS = Fe + S.

using UnityEngine;

using System.Collections;

public class fe\_behaviour : MonoBehaviour

{

bool move\_body = false;

bool fly = false;

float speed;

void FixedUpdate()

{

move\_body = main\_gui.move\_body;

speed = main\_gui.speed;

if (move\_body)

{

if (!fly)

{

Vector3 force\_vector = Quaternion.AngleAxis(-1 \* Random.Range(25, 45), Vector3.forward) \* Vector3.right;

//force\_vector = Quaternion.Euler(0, 45, 0) \* force\_vector;

rigidbody.AddForce(speed \* force\_vector \* 100);

fly = true;

}

}

}

}

C#

Данный скрипт используется для симуляции поведения Fe. Так, как тип этой реакции – разложение, значит мы должны придать Fe определенную энергию, для разрыва связи FeS.

Вначале мы создаем переменную, которая будет сообщать скрипту начать ли движение. Значение данной переменной меняется, только если мы нажмем на кнопку “Начать”. Так как функция FixedUpdate выполняется несколько раз за секунду, значит просто вызвать функцию AddForce не получится, ведь данная функция придает силу твердому телу, выполняя эту функцию много раз за секунду, мы получим огромную скорость у нашего тела, которая будет расти с увеличением времени с запуска программы. Значит нам нужно вызвать данную функцию лишь один, поэтому мы создаем переменную “fly”, указывая ей начальное значение “false”. После этого мы проверяем равно ли нашое текущее значение “false”, если да, то вызываем функцию AddForce, и обязательно меняем значение переменной “fly” с “false” на “true”.

В качестве аргументов данная функция принимает вектор силы, а также необязательный параметр для типа силы. Так, как наш атом должен вылететь под углом мы должны посчитать вектор силы. В итоге мы получаем следующую формулу:

В которой , мы наклоняем вектор имееющий координаты (0,0,1). Таким образом мы наклоняем наш вектор на случайный угол в границах от 25 до 45 градусов и применяем силу к телу по данному вектору.

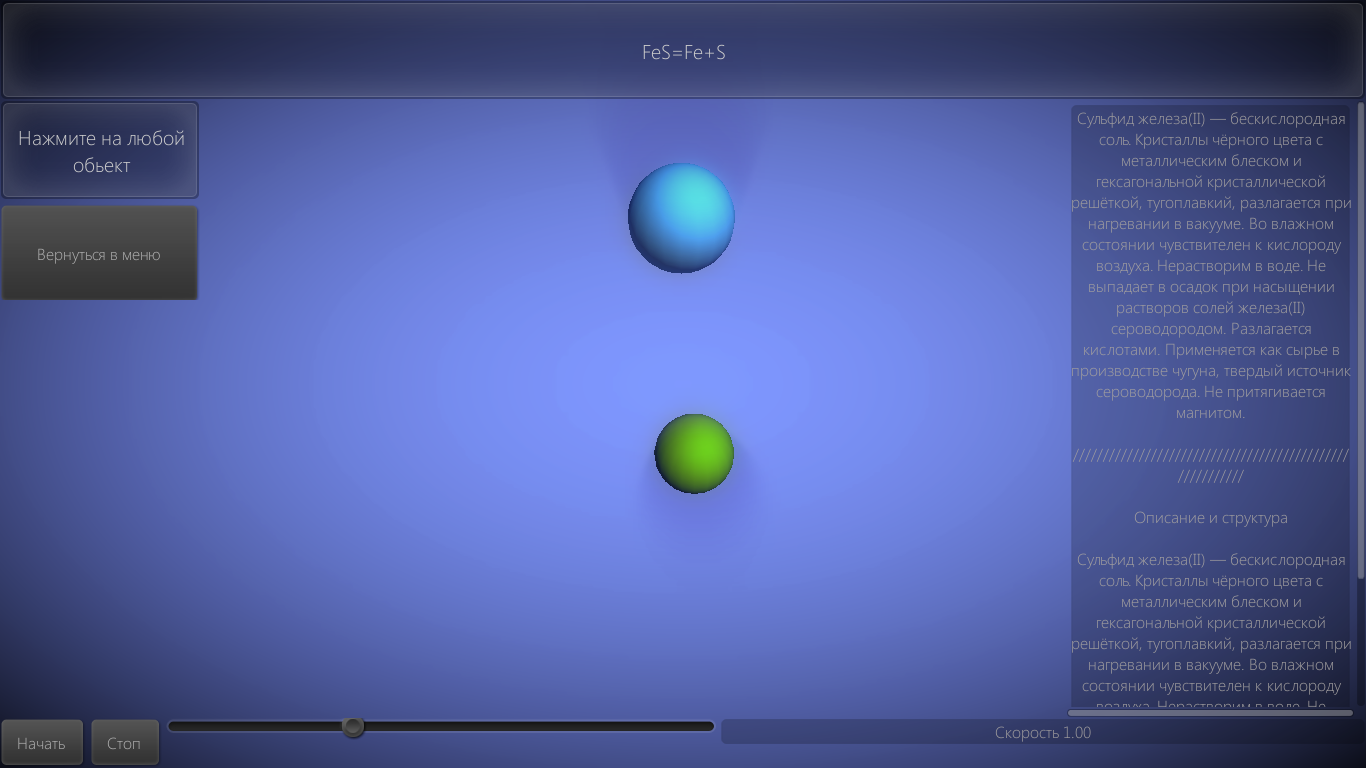


Рис 6. Траектория Fe

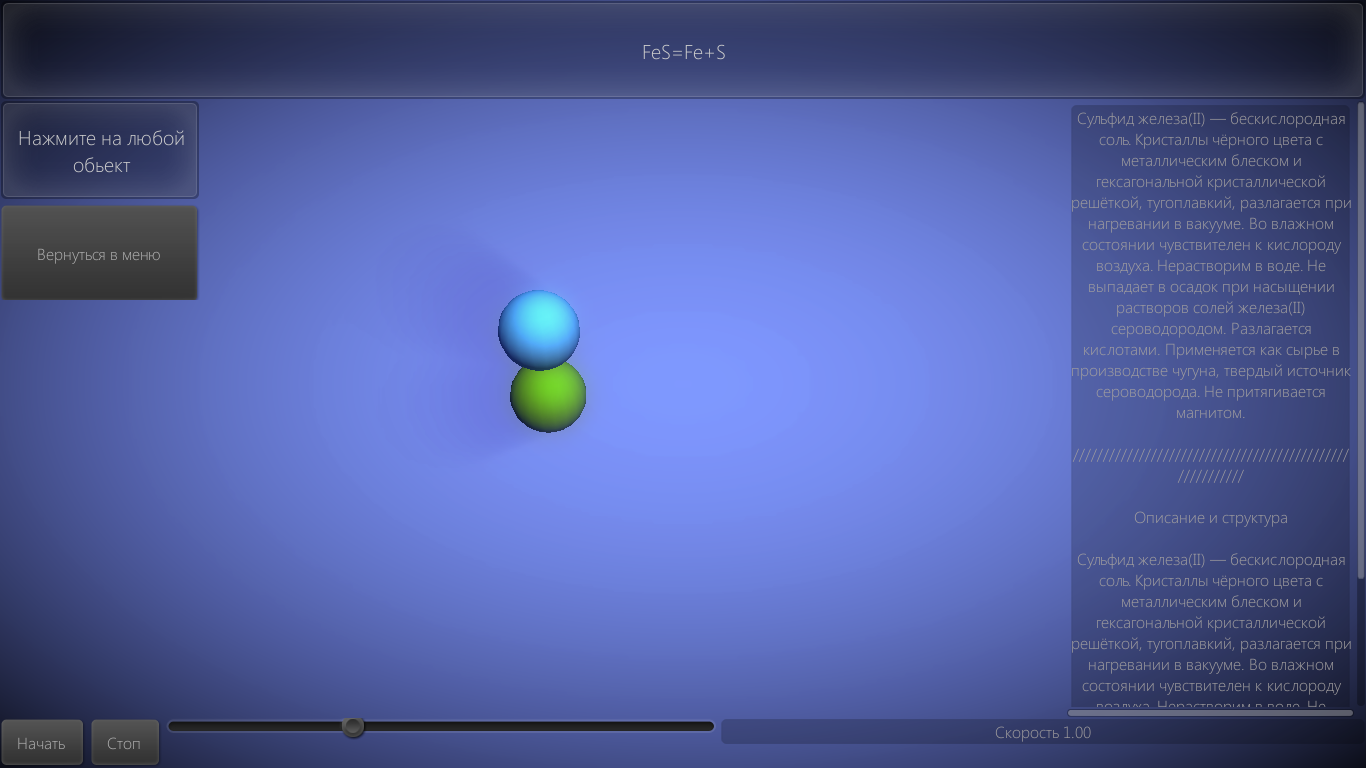


Рис 7. Интерфейс режима симуляции

В режиме симуляции есть свой пользовательский интерфейс, который состоит из:

* Блока, где выводится формула текущей реакции
* Блок, для вывода текущего вещества
* Блок, где выводится основная информация о реакции
* Кнопки, для управления реакцией

Информация о реакции получается из внешнего сервера. Код получения информации:

using UnityEngine;

using System.Collections;

public class test\_get\_info : MonoBehaviour

{

bool is\_show = false;

public string url;

public string text = "I get some info from server";

Vector2 scroll\_position;

public GUISkin style;

IEnumerator Start()

{

WWW www = new WWW(url);

yield return www;

text = www.text;

}

void OnGUI()

{

GUI.skin = style;

GUILayout.BeginArea(new Rect(Screen.width - 300, 100, 300, Screen.height - 150));

scroll\_position = GUILayout.BeginScrollView(scroll\_position, true, true, GUILayout.Width(300));

//

GUILayout.Label(text);

//

GUILayout.EndScrollView();

GUILayout.EndArea();

}

}

C#

Вначале мы создаем переменную, где будем хранить адрес получения информации. После создаем строку, для хранения полученного текста. В функции “Start” мы создаем экземпляр класса WWW, указывая наш адрес. Теперь вся информация с сервера хранится в этой переменной. Она может хранить видео, фотографии, аудио и конечно же текст.

В функции “ OnGUI ” мы создаем контейнер, внутри которого создаем элемент ScrollView, для создания прокрутки нашего текста.

Текст хранится в виде файла на сервере github

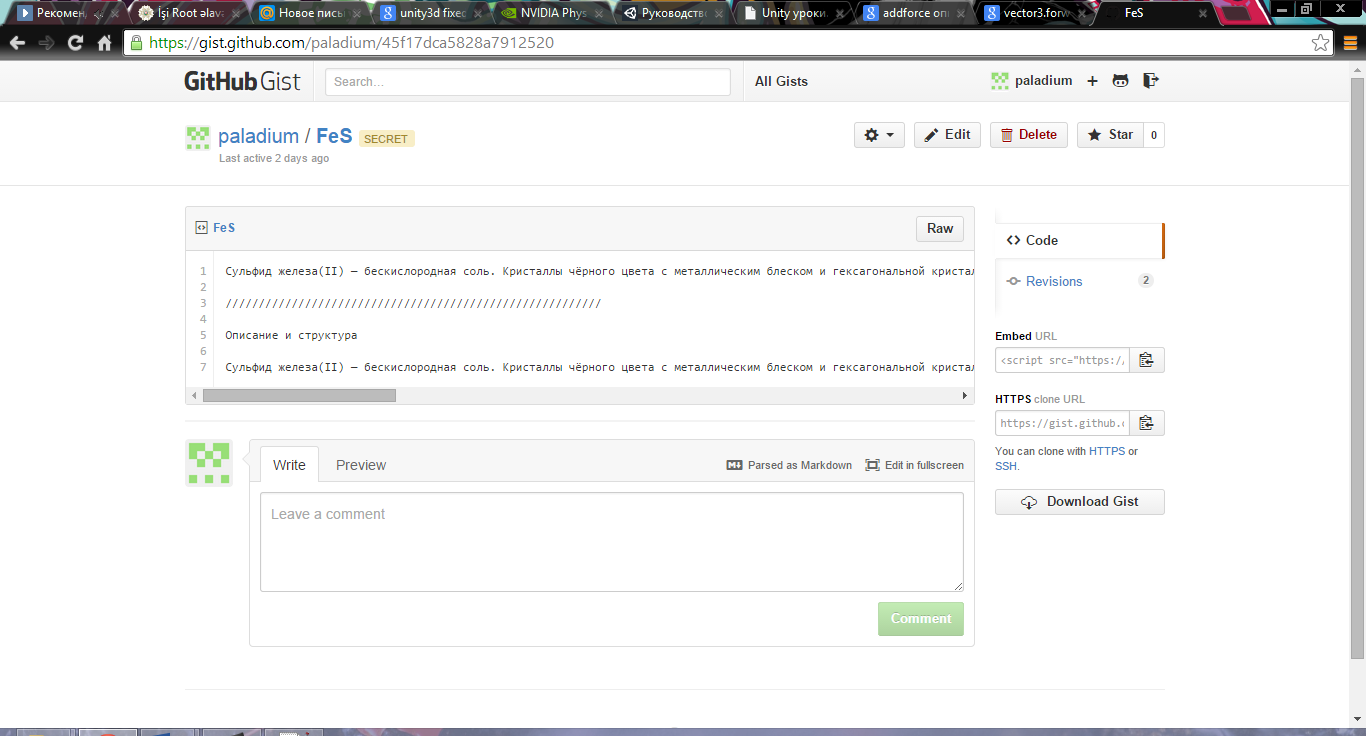


Рис 8. Текст на сервере github

# Видеопроигрыватель

В моей программе реализован видеопроигрыватель, позволяющий пользователю просмотреть видео об интересных химических реакциях.



Рис 9. Интерфейс видеопроигрывателя

Часть кода на видеопроигрыватель:

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class menu : MonoBehaviour

{

…

public List<MovieTexture> videos = new List<MovieTexture>();

public List<AudioClip> audios = new List<AudioClip>();

….

bool is\_show\_video = false;

…

if (is\_show\_video)

{

GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), videos[current\_video], ScaleMode.ScaleAndCrop);

GUILayout.BeginArea(new Rect(50, Screen.height - 100, Screen.width, 100));

GUILayout.BeginHorizontal();

if (GUILayout.Button("Старт", GUILayout.Height(100), GUILayout.Width(150)) && videos[current\_video].isReadyToPlay)

{

videos[current\_video].Play();

audio.clip = audios[current\_video];

audio.Play();

}

GUILayout.FlexibleSpace();

…….

if (GUILayout.Button("Предыдущее видео", GUILayout.Height(100), GUILayout.Width(150)) && current\_video > 1)

{

videos[current\_video].Stop();

current\_video--;

audio.clip = audios[current\_video];

}

GUILayout.FlexibleSpace();

if (GUILayout.Button("Следующее видео", GUILayout.Height(100), GUILayout.Width(150)) && current\_video < videos.Count - 1)

{

videos[current\_video].Stop();

current\_video++;

audio.clip = audios[current\_video];

}

GUILayout.FlexibleSpace();

GUILayout.EndHorizontal();

GUILayout.EndArea();

….

}

C#

Вначале мы создаем динамический массив, для хранения отдельно звука и видео. Далее мы отображаем наше видео с помощью функции GUI.DrawTexture().

После этого мы рисуем наши кнопки и пишем для каждой кнопки свою логику. К примеру для кнопки Старт логика следующая – надо синхронно запустить текущее видео и звук к текущему видео.

Для переключение между видеозаписями созданы специальные кнопки, которые меняют номер активного видео, а также текущее аудио. Видео хранится в формате .ogg, что позволяет очень сильно оптимизировать программу, а также снизить нагрузку на процессор.

# Теория

В разделе теория можно прочитать основную информацию об химических реакциях.

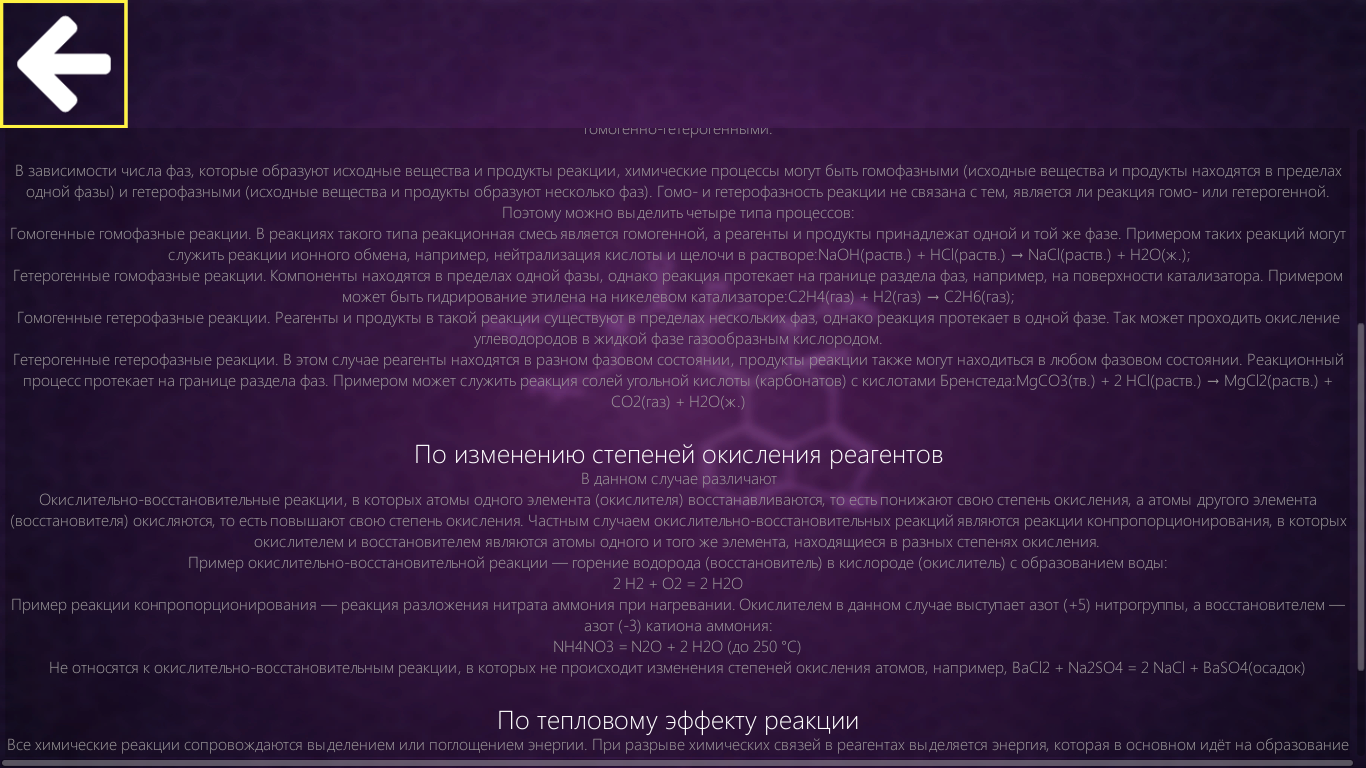


Рис 10. Раздел теория

# Выводы

Были изучены различные виды химических реакций, их свойства, особенности и характеристики.

Для большего понимания данного материала была создана компьютерная программа, в которой было визуализировано разные виды реакций. Улучшились знания языка c#, а также были углублены знания в программах:

* Unity3D
* Adobe Photoshop
* Autodesk 3Ds Max