1. Принципы дискретизации и квантования для оцифровки мультимедийной информации.

Дискретизация – это разделение пространства или времени на фиксированные по размеру области (точки, которые точками, по сути, не являются) или отрезки. Так описываемое двумерное изображение разбивается на маленькие плоскости. **В пределах каждой такой плоскости характеристики изображения считаются одинаковыми.** Понятно, что при этом часть информации теряется. Мы не получаем точную копию реального объекта, мы лишь описываем его существенные характеристики.

Разделение непрерывного ряда значений какой-либо характеристики на ограниченное количество диапазонов называют **квантованием**. В компьютере сохраняется лишь номер диапазона, в который попало конкретное значение свойства.

Если при дискретизации разделяется время или пространство, то при квантовании этому подвергаются возможные значения свойств.

Понятно, что чем более дискретна и квантована естественная информация, тем более точно она сохранена в памяти компьютера. Однако этой памяти потребуется больше.

Человеческие органы чувств имеют свои ограничения. Поэтому различие в цвете двух точек мы можем не уловить, хотя их физические характеристики длин волн могут различаться. Поэтому в определенных значениях потеря информации может быть незаметна для человека.

Общие принципы оцифровки мультимедийной информации – этапы дискретизации и квантования (на примере оцифровки звуковой волны).

Мультимедийная информация – графическая и звуковая, воспринимается органами чувств человека только в аналоговом виде и только в определенных пределах по графике, звуку и скорости воспроизведения чередующихся кадров в видео или анимации.

Это обстоятельство лежит в основе принципов оцифровки мультимедийной информации и записи ее в графические, аудио и видео-файлы.

Оцифровка мультимедийной информации подразумевает запись в цифровой форме в файлы соответствующего формата для хранения и воспроизведения на компьютере аналоговой информации

Теорема Котельникова

Следствие из теоремы В. Котельникова (фундаментального утверждения в области цифровой обработки сигналов, связывающего непрерывные и дискретные сигналы):

«Любой аналоговый сигнал может быть восстановлен с какой угодно точностью по своим дискретным отсчётам, взятым с частотой f > 2fm , где fm — максимальная частота, которая ограничена спектром реального сигнала».

Поэтому частота дискретизации звука должна быть не менее 40 кГц, поскольку максимальная частота звука для восприятия человеком составляет примерно 20 кГц.

ОЦИФРОВКА ЗВУКА (дискретизация и квантование)

При оцифровке звука (Digital sampling) при выполнении этих этапов звуковой платой компьютера, как правило, для них назначаются параметры в следующих пределах:

- **Дискретизация** (Sampling rate) – от 11 кГц до 44 кГц
- **Квантование** (Sampling size) – от 8 бит (речь) до 16 бит (музыка)

Объем звукового файла:

V(байт)=Sr(Гц) * Ss(бит)/8 * t(c) * n

Типы звуковых форматов: **WAV** (несжатый), **MP3** (сжатый по определенному алгоритму) и др.

2. Общая структура форматов сжатия мультимедийной информации: JPEG для графики и MPEG для видео.

JPEG

Название алгоритма компрессии — аббревиатура от Joint Photographic Expert Group, инициативной группы, образованной из экспертов ITU (International Telecommunication Union) и ISO (International Organization for Standartization). Именно поэтому в ее названии присутствует приставка Joint. В 1992 г. JPEG был объявлен международным стандартом в области графических изображений.

При компрессии методом JPEG качество теряется всегда. При этом всегда есть выбор: отдать предпочтение качеству в ущерб объему (размер файла сожмется приблизительно в три раза) или же наоборот, добиться минимального размера изображения, при котором оно еще останется узнаваемым (степень компрессии может достигать 100). Сжатие, при котором различие в качестве между получившимся

изображением и оригиналом еще остается незаметным, дает 10-20-кратное сокращение размера файла.

Как происходит сжатие

1. Первый этап заключается в конвертировании цветовой модели изображения (обычно RGB) в модель, где яркостная и цветовая составляющие разнесены (например, YCbCr или YUV), что позволяет оптимально подойти к выбору степеней компрессии для каждого канала (с учетом особенностей восприятия глазом).

2. На следующем этапе происходит т. н. пре фильтрация, при которой соседние пиксели отдельно в каждом из каналов Cb и Cr группируются попарно в горизонтальном и вертикальном направлениях, а яркостный канал Y оставляется без изменений. После этого вся группа из четырех пикселов получает усредненное значение соответствующих компонент Cb и Cr.

3. Полученная информация, прошедшая стадию первичной «очистки», отдельно в каждом канале снова группируется в блоки, но уже размером 8x8, после чего для них применяется основное сжатие — т. н. дискретное косинусное преобразование. В результате информация о распределении яркости пикселов преобразуется в другой вид, где она описывается распределением, основанном на частоте появления той или иной яркости пикселов.

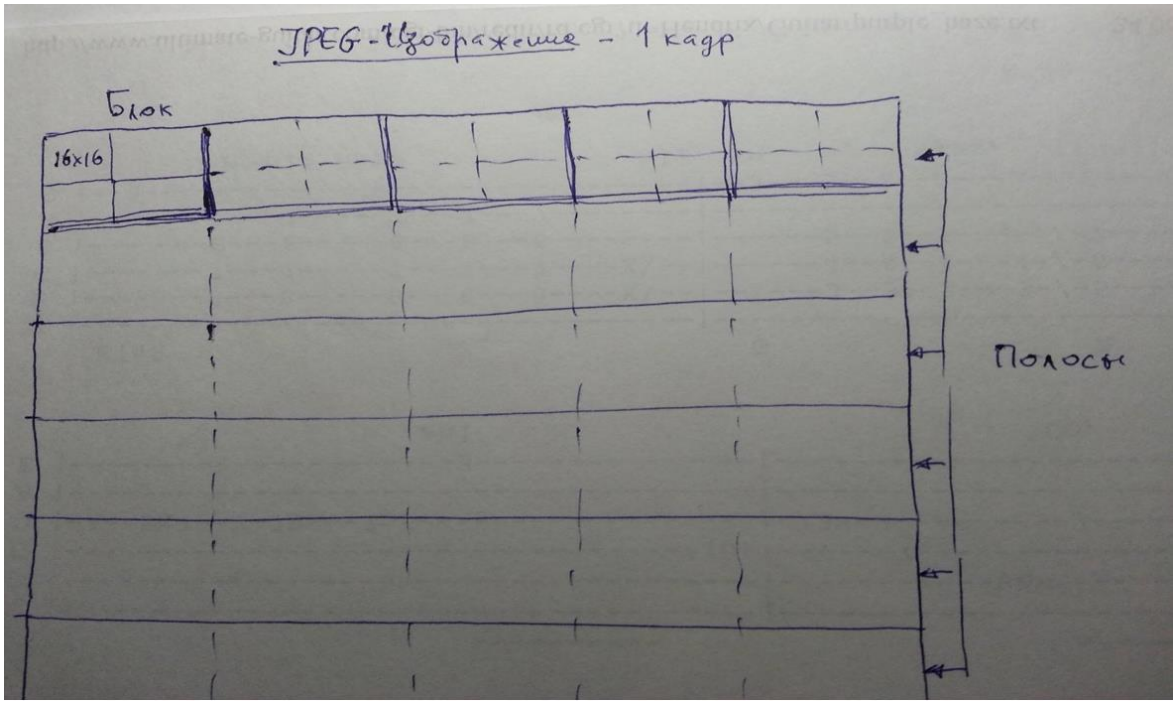
4. Следующий этап — удаление малозаметной глазу информации из блока, или квантование (quantization). Все составляющие делятся на различные коэффициенты, определяющие значимость каждой из них для качественного восстановления исходного изображения, и результат округляется до целого значения. Именно эта процедура вносит наибольшие потери качества, снижая конечный объем изображения. Высокочастотные составляющие квантуются грубо, а низкочастотные — точнее, поскольку наиболее заметны. Дабы несколько сгладить снижение качества, в канале яркости используются меньшие коэффициенты деления, чем в каналах цветности. В результате квантования получается набор составляющих, по которым исходное изображение восстанавливается с заданной точностью

5. После выполнения основной работы по сжатию изображения дальнейшие преобразования сводятся к второстепенным задачам: оставшиеся составляющие собираются в последовательность таким образом, чтобы сначала располагались отвечающие за крупные детали, а потом — за все более мелкие. Затем получившаяся последовательность сжимается: сначала обычным RLE, затем методом Хаффмана.

6. И наконец, чисто техническая стадия — данные заключаются в оболочку, снабжаются заголовком, в котором указываются все параметры компрессии с тем, чтобы изображение можно было восстановить. Впрочем, иногда в заголовки не включают эту информацию, что дает дополнительный выигрыш в компрессии, однако в этом случае нужно быть уверенным, что приложение, которое будет читать файл, о них знает.

Недостатки JPEG

- Невозможность достичь высоких степеней сжатия за счет ограничения на размер блока (только 8x8).
- Изображение нельзя отобразить до тех пор, пока оно не загрузится полностью.
- Поддерживаются только RGB-изображения
- Закругление острых углов и размывание тонких элементов в изображении.



MPEG

Стандарт сжатия MPEG разработан Экспертной группой кинематографии (Moving Picture Experts Group - MPEG). MPEG это стандарт на сжатие звуковых и видео файлов в более удобный для загрузки или пересылки, например через интернет, формат. Существуют разные стандарты MPEG (как их еще иногда называют фазы - phase): MPEG-1, MPEG-2, MPEG-3, MPEG-4, MPEG-7.

Как MPEG работает

В зависимости от некоторых причин каждый frame (кадр) в MPEG может быть следующего вида:

- **I** (Intra) frame - кодируется как обыкновенная картинка.
- **P** (Predicted) frame - при кодировании используется информация от предыдущих I или P кадров.
- **B** (Bidirectional) frame - при кодировании используется информация от одного или двух I или P кадров

Последовательность кадров может быть например такая: IBBPBBPBBPBBIBBPBBPB…

Форматы сжатия семейства MPEG сокращают объем информации следующим образом:

- Устраняется временная избыточность видео (учитывается только разностная информация).
- Устраняется пространственная избыточность изображений путем подавления мелких деталей сцены. · Устраняется часть информации о цветности.
- Повышается информационная плотность результирующего цифрового потока путем выбора оптимального математического кода для его описания.

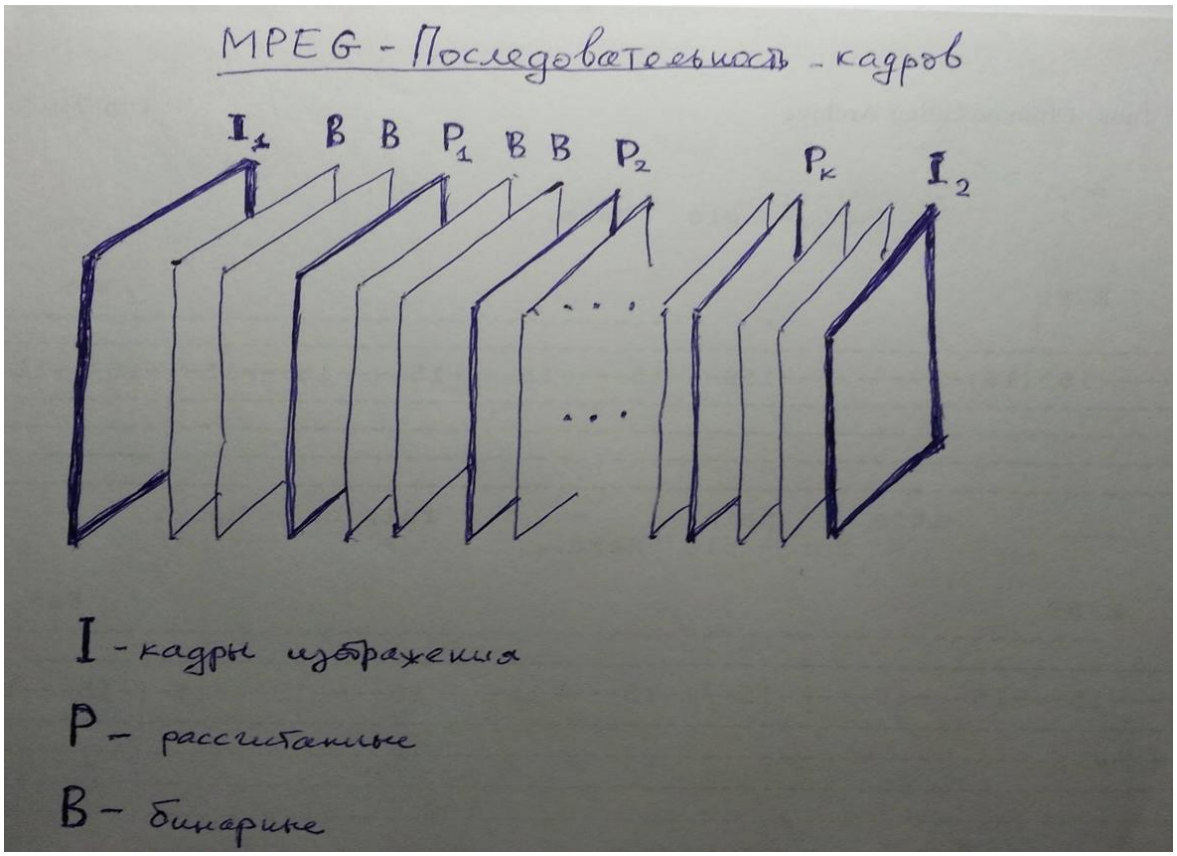
Форматы сжатия MPEG сжимают только опорные кадры – **I**-кадры (Intra frame – внутренний кадр). В промежутки между ними включаются кадры, содержащие только изменения между двумя соседними **I**-кадрами – **P**-кадры (Predicted frame – прогнозируемый кадр). Для того чтобы сократить потери информации между **I**-кадром и **P**-кадром, вводятся так называемые **B**-кадры (Bidirectional frame – двунаправленный кадр). В них содержится информация, которая берется из предшествующего и последующего кадров. При кодировании в форматах сжатия MPEG формируется цепочка кадров разных типов.

Чтобы декодер мог работать, необходимо, чтобы первый P-кадр в потоке встретился до первого **B**, поэтому сжатый поток выглядит так:

0 x x 3 1 2 6 4 5 ...

где числа — это номера кадров, а xx может не быть ничем, если это начало последовательности, или **B**-кадры -2 и -1, если это фрагмент из середины потока.

Сначала необходимо раскодировать **I**-кадр, затем **P**, затем, имея их оба в памяти, раскодировать **B**. Во время декодирования **P** показывается **I**-кадр, **B** показывается сразу, а раскодированный **P** показывается во время декодирования следующего.



3. Внедрение звука в анимационный клип, типы синхронизации звука в среде Adobe Animate.

Способы загрузки звука:

1. Загрузить внешний звуковой файл в ключевой кадр

2. Встроить звук в swf-файл в процессе его создания

3. Получить аудио-ввод с помощью микрофона

4. Получить потоковые данные, передаваемые с сервера

5. Работать с аудиоданными, создаваемыми динамически.

Звук внедряется в анимационный клип посредством программного кода или сами, перенесением звука на слой. Во втором варианте мы можем взаимодействовать со звуком таким образом, что можем изменить тип его синхронизации, наложить эффект(увеличение громкости, левый, правый канал), выбрать количество циклов воспроизведения.

Типы синхронизации звука в Adobe Animate:

Событийный

·Событие (Event) – событийный звук (по умолчанию).

· Начать (Start) – запуск событийного звука.

· Остановить (Stop) – остановка событийного звука.

Потоковый

· Поток (Stream) – потоковый звук для синхронизации анимации со звуком.

Событийный звук Событие (Event)

· Воспроизведение начинается при входе в ключевой кадр, содержащий звук, и не зависит от дальнейших кадров монтажной линейки.

· Воспроизведение начинается после того, как звук полностью загрузится в память.

· Событийные звуки накладываются (микшируются), если ролик зациклен.

· Зацикливание событийного звука можно использовать для создания фонового сопровождения.

Начать (Start)

· Применяется для исключения наложения событийных звуков самих на себя.

· Воспроизводится только, если другой экземпляр того же звука не воспроизводится в данный момент.

· Для остановки звука нужно сделать ключевой кадр с играющим в «данный» момент звуком и поставить тип воспроизведения Событие -> Остановить.

Поток (Stream)

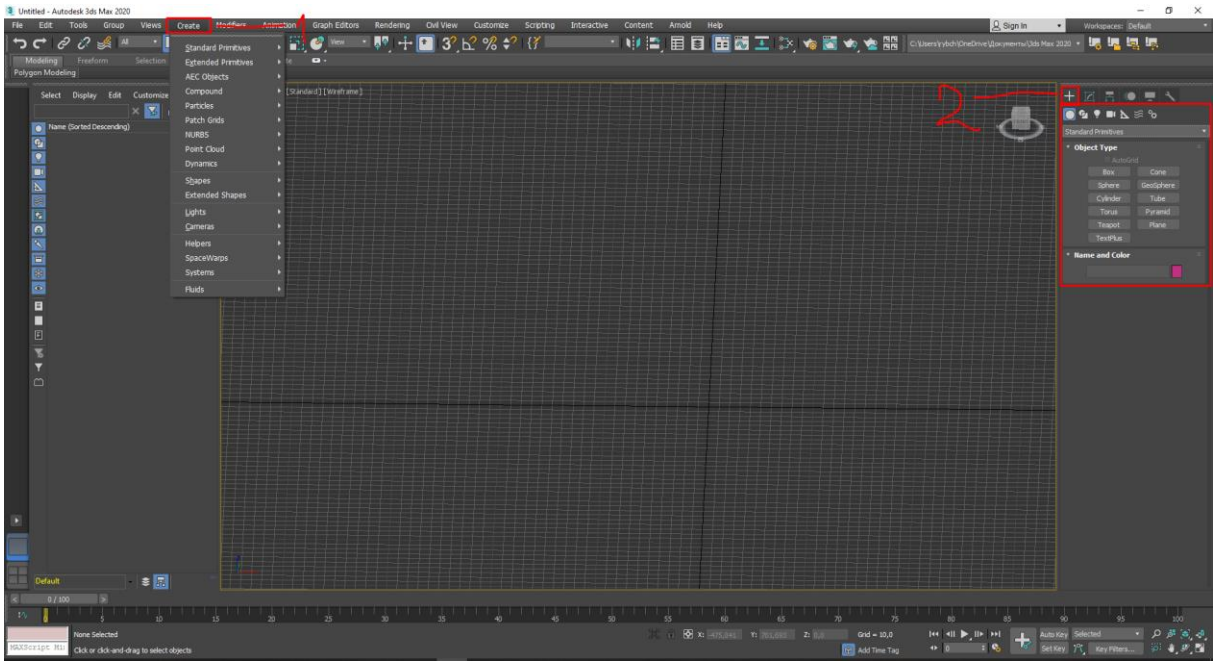
· Звук воспроизводится только в содержащих его кадрах – зависит от содержащей его монтажной линейки.

· Звук имеет приоритет над визуальным содержимым кадров монтажной линейки.

· Звук разбивается на отдельные фрагменты, которые жестко привязываются к кадрам монтажной линейки.

· Звук типа Поток не рекомендуется зацикливать. Для кнопок используются ТОЛЬКО событийные звуки.

4. Создание и редактирование объектов в среде 3ds MAX. Использование модификаторов.

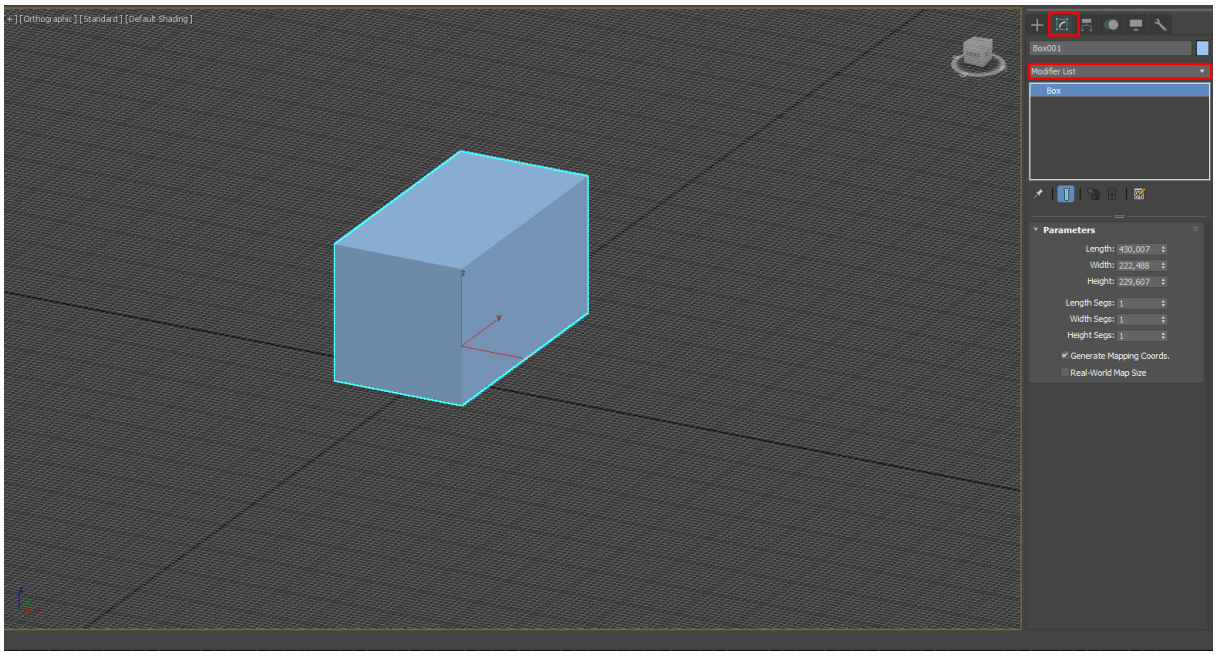


Два варианта создания объектов:

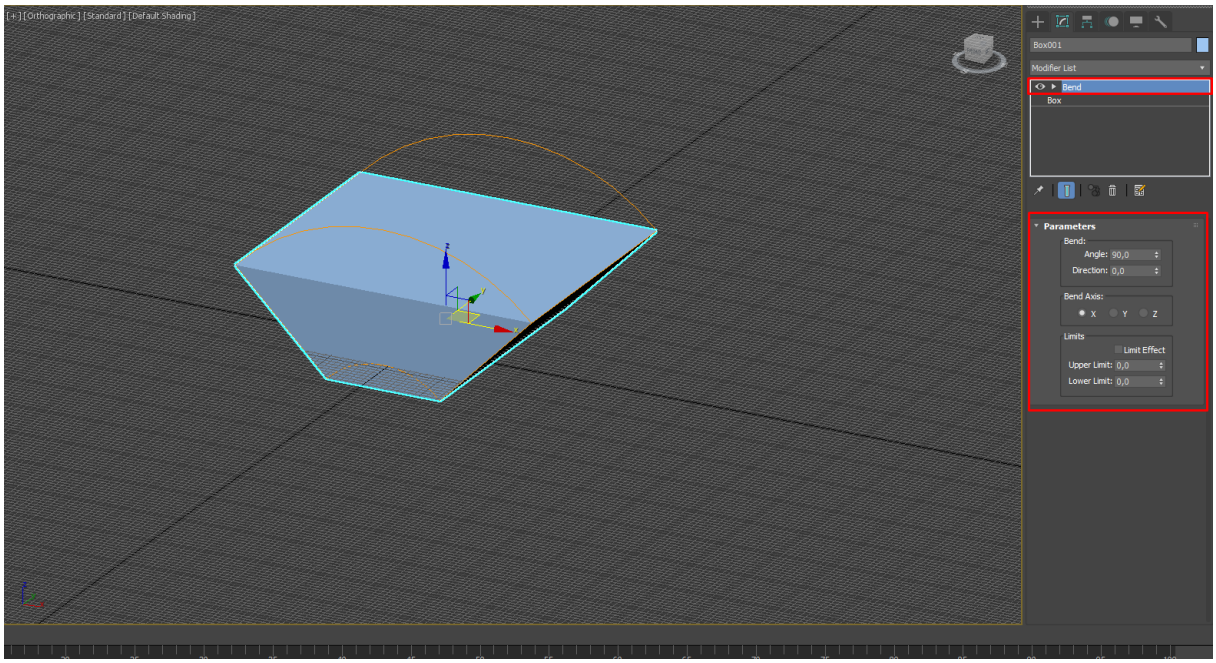
1 - через меню Create

2 - через command панель

Затем для применения модификаторов требуется в панели command выбрать 2ую вкладку и кликаем на список Modificator list.



После чего выбираем один из модификаторов, например Bend, и внизу меняем параметры модификатора.



5. Использование освещения, материалов и текстур в среде 3ds MAX.

Освещение:

Для освещения сцены используются специальные объекты из вкладки Create - Lights

Target Spot и Target Direct генерируют направленный свет, могут быть использованы для имитации света от ламп, фонариков, автомобильных фар и др.

У Target Spot - свет расходится по форме конуса, а у Target Direct - свет распространяется параллельно, по форме параллелепипеда.

Слово target означает, что направлением света можно управлять специальным маркером в виде небольшого кубика, источники Free Spot и Free Direct его не имеют.

Omni - точечный источник рассеянного света, распространяемого во всех направлениях. Используется для имитации света обычной лампочки или для общего освещения.

Пример использования:

1) Расположите в сцене любой объект и создайте под ним плоскость, которая будет принимать тени.

2) Теперь установите Target Spot

3) Для основного источника света необходимо включить параметр Shadows (тени), что делает картинку реалистичней.

4) Параметр Multiplier во вкладке Intenity/Color/Attenuation влияет на силу источника

света

-Параметр Dens во вкладке Shadow Parameters влияет на плотность теней

5) Для того чтобы граница света-тени была мягкой, во вкладке Spotlight Parameters поставьте значение Hotspot/Beam в несколько раз меньше значения Falloff/Field, изменения вы будете наблюдать в окне проекции

)

Текстурирование:

Пример использования текстур и материалов(в вопросе стоит “Использование…”)

1) Создайте несколько примитивов: бокс и шар(сферу)

2)Откройте редактор материалов, нажав клавишу «M», и создайте новый материал. Назначьте в слот «Diffuse» карту «Checker», выбрав ее в свитке «standart» перечня карт.

3) Выберите бокс. Примените к нему модификатор «UVW Map», выбрав его из списка.

4) В разделе «Mapping» ставим точку возле «Box»— текстура корректно расположилась по поверхности.

— Ниже задаются размеры текстуры или шаг повторения ее рисунка. В нашем случае регулируется повторение рисунка, так как карта Checker — процедурная, а не растровая.

5) Желтый прямоугольник, обрамляющий наш объект, — это «гизмо», область в которой воздействует модификатор. Ее можно перемещать, вращать, масштабировать, центрировать, привязывать к осям. С помощью гизмо текстура помещается в нужное место.

6) Пример для сферы!!!!!!!!!!! Выберите сферу и присвойте ей модификатор «UVW Map».

— В разделе «Mapping» установите точку напротив «Sperical». Текстура приняла форму шара. Чтобы это было лучше видно увеличьте шаг клетки. Параметры гизмо не отличаются от бокса, кроме того, что гизмо шара будет иметь соответственно сферическую форму.

6. Методы редактирования поверхностей 3D-объектов в среде 3ds MAX.

При моделировании на основе примитивов выполняются **2 этапа**:

1) создание исходного объекта на основе примитива

2) модификация объекта

Модификация исходного объекта производится в основном следующими средствами:

- командами редактора во вкладке ***Modify***
- при помощи **булевых операций**;
- с помощью параметров на основе ***модификаторов***,
- либо с помощью ***полигонального моделирования***

Модификаторы — это инструменты, с помощью которых можно "обрабатывать" трехмерные объекты.

Модификаторы применяются для изменения формы объектов, деформирования поверхности и т. п. с настройкой собственных параметров, которыми можно управлять степенью воздействия этих модификаторов на объект.

Все применяемые к объекту модификаторы заносятся в список панели ***Modifier List*** (стек модификаторов) вкладки Modify панели Command.

1) Modify:

Lofting **-** создание объектов на основе сечений

Объект на основе сечения *Loft* — это трехмерный объект, поверхность которого создана как огибающая плоских опорных форм, размещенных вдоль некоторого пути.

Формы, на которые опирается поверхность подобного объекта, рассматриваются как его поперечные сечения. Путь определяет размещение сечения в пределах объекта.

Чтобы создать объект, основанный на сечениях, необходимо:

- создать две сплайновых формы: одна для сечения, вторая для пути;
- выделить сплайн для пути (направляющей);
- выполнить команду Create/Geometry/CompoundObjects/Loft щелкнуть на кнопке GetShape (Получить форму) и выделить в окне проекции сплайн для сечения.

Для редактирования опорного сечения необходимо перейти к панели *Modify/Deformations* и выбрать команду *Scale.*

2) Булевские операции: объединение, вычитание, пересечение.

- создать два объекта на основе стандартных примитивов;
- наложить объекты друг на друга перемещением;
- выполнить команду Compound Objects и выбрать в ней режим Boolean
- в окне параметров команды нажать кнопку Pick Operand B (исходный выделенный объект имеет имя A)
- выделить другой объект (объект B) – получится составной объект: из объекта A будет вырезана часть от наложения на него объекта B (по умолчанию работает логическая операция вычитания Substraction).
- можно примените и другие логические операции: объединения Union, пересечения Intersection.

3) Lathe - создание поверхностей вращения

Чтобы создать трехмерное тело методом вращения профиля, необходимо:

- нарисовать двухмерную форму - профиль, который должен представлять собой одну зеркальную половину поперечного сечения будущего тела вращения.
- применить к ней модификатор Lathe (Вращение), который строит трехмерное тело, выполняя полный или неполный оборот формы относительно одной из трех координатных осей.

4) Extrude и Bevel - методы «выдавливания»

Модификаторы Extrude (Выдавливание) и Bevel (Выдавливание со скосом), которые схожи по своему действию и применяются к любой сплайновой форме.

Главной настройкой модификаторов Extrude и Bevel является амплитуда выдавливания. Для модификатора Bevel— это параметр Height (Высота), а для Extrude— Amount (Величина). Величину скоса задает параметр Outline (Масштаб).

5) Модификаторы для сглаживания поверхности:

Поверхность любого 3D-объекта представляет собой сетку (mesh) из плоских треугольников, которые объединяются в полигоны и составляют в целом каркас трехмерного объекта.

Для сглаживания поверхности полигональной модели применяются специальные модификаторы – MeshSmooth (Сглаживание сетки) и TurboSmooth (Турбо Сглаживание).

•MeshSmooth (Сглаживание сетки) — сглаживает поверхность добавлением дополнительных граней вдоль тех ребер и вершин, где есть резкие переходы от одного полигона к другому. Сглаживание можно применить ко всей сетке установкой флажка Apply to Whole Mesh (Применить ко всей сетке).

• TurboSmooth (Турбосглаживание) работает быстрее модификатора MeshSmooth и лучше сглаживает поверхности, но добавляет значительно больше граней и полигонов.

6) Взаимодействия с полигонами и их составляющими.

Для начала необходимо применить к объекту модификатор Edit Poly. Для этого нужно перейти на вкладку Modify – Modifier List – Edit Poly. Я рекомендую использовать именно его, потому что при желании его свободно можно удалить со всеми внесенными изменениями. Если вы превратите объект в Editable Poly, то вернуть объект к начальному виду будет сложнее.

Для того, чтобы выбирать подобъекты, необходимо включить редактирование точек (Vertex), ребер (Edge), краев (Border), полигонов (Polygons) или элементов (Elements) в разделе Selection. После этого понадобится только нажать на те подобъекты, которые нужно менять.

7. Назначение и принципы использования компонент Physics, Collider и Materials в среде Unity.

Компоненты **Physics** используется для симуляции реалистичной физики. Для обработки столкновений у объекта должен быть компонент Collider. Основной компонент для симуляции физики это Rigidbody. После применения компонента **Rigidbody** на объект начнет воздействовать сила гравитации и он начнет падать. Если у объекта есть компонент Collider то будут обрабатывается столкновения объекта с другим объектами.

Компоненты **Colliders** определяют форму объекта для физических столкновений. Коллайдер не обязательно должен точно соответствовать объекту главное чтобы он описал его форму. Базовые формы коллайдера это **Box, Sphere, Mesh, Capsule**. С помощью Mesh Collider можно указать произвольную форму коллайдера.

Компонент **Materials** используется для симуляции реалистичной поверхности объектов. В этом компоненте можно задать текстуру поверхности, ее цвет, прозрачность, шероховатость, металличность.

8. Организация движения, вращения и масштабирования объектов в среде Unity на языке C#. Использование методов Rotate() и Translate().

В юнити есть физическое и нефизическое **движение**.

Нефизическое:

transform.position = new Vector3(…) - перемещение объекта

transform.Translate(…) - учитывает направление объекта

transform.Lerp(…) – линейная интерполяция между двумя точками.

Физическое:

для физических - взаимодействие со скоростью(velocity) - fixedUpdate

addforce - сообщение каждую секунду силы(ускорение)

velocity - добавление скорости напрямую объекту

//должен быть в Update()

```
te                          void                          KeyMoving

    m_dir    =    new    Vector3(Input.GetAxis("Vertical"),    0,
GetAxis("Horizontal"));
nsform.Translate(m_dir       *       4f       *       Time.deltaTime
```

Вращение

.Rotate(…) - метод для вращения по градусам

.rotation = … - свойство отвечающее за вращение по градусам

.RotateAround(…) - метод для вращения вокруг объекта

Quaternion() - вращение задается один угол поворота и ось вращения

```
form.eulerAngles += new Vector3(1,0,1);
```

Для поворота в 3D-пространстве используются углы Эйлера в свойстве **eulerAngles** для компоненты transform:

```
                    angl                        =
pdate ()
=5.0f; transform.eulerAngles=new Vector3(angl,0,0); }
```

Более просто для вращения можно использовать функцию **Rotate**(), которая автоматизирует процесс приращения угла поворота, заданного значением угла для соответственно осей X, Y, Z:

```
   void Update () { transform.Rotate(3, 0, 0);}
```

Для измерения угла поворота в Unity используются градусы

В поворотах с помощью функции Rotate или свойства eulerAngles необходимо задавать в общем случае три угла поворота вокруг осей X, Y и Z, при этом обратное вращение для них будет некоммутативной (неперестановочной) операцией.

При задании вращения с помощью **кватерниона** (Quaternion) задается один угол поворота и ось вращения в трехмерном пространстве, что позволяет полностью восстановить первоначальное положение объекта после его поворота.

Кроме того, помощью кватернионов можно осуществить более гладкую интерполяцию между двумя разными положениями объекта, чем это делают углы Эйлера и функция Rotate.

Кватернион – это гиперкомплексное число, представляемое четверкой чисел q=(w,x,y,z) или в виде q=[w,v], где: w – число, а v=(x,y,z) – вектор в трехмерном пространстве.

```
rnion                                                    ori
                                                         ang
                                                       Start
{          orig      =      transform.rotation;
                                                      Update
{                        angl            +=             3.0
Quaternion  rotY  =  Quaternion.AngleAxis(angl,  Vector3.up
transform.rotation = orig * rotY;
}

жен               быть               в               Update
       inp         =         Input.GetAxis("Mouse         Y"
   var   rotCamera   =   new   Vector3(-inp   *   _speedRotate
deltaTime,                        0,                        0
  _camera.Rotate(rotCamera);
гие
sform.RotateAround(_stoveTransform.position, Vector3.up, 2
put.GetAxis("Horizontal") * Time.deltaTime);
```

Масштабирование

localScale

```
pdate ()
sform.localScale += new Vector3(0.1f, 0.0f, 0.0f);}
```

9. Понятие о кватернионах, использование кватернионов для программирования поворотов 3D-объектов.

В поворотах с помощью функции Rotate или свойства eulerAngles необходимо задавать в общем случае три угла поворота вокруг осей X, Y и Z, при этом обратное вращение для них будет некоммутативной (неперестановочной) операцией.

При задании вращения с помощью кватерниона (Quaternion) задается один угол поворота и ось вращения в трехмерном пространстве, что позволяет полностью восстановить первоначальное положение объекта после его поворота.

Кроме того, с помощью кватернионов можно осуществить более гладкую интерполяцию между двумя разными положениями объекта, чем это делают углы Эйлера и функция Rotate.

Кватернион – это гиперкомплексное число, представляемое четверкой чисел q=(w,x,y,z) или в виде q=[w,v], где: w – число, а v=(x,y,z) – вектор в трехмерном пространстве.

Поворот с использованием кватернионов

Поворот с использованием кватернионов использует конструкцию Quaternion.AngleAxis с двумя аргументами, где первый – число, соответствующее углу поворота в градусах, а второй – Vector3, указывающий для объекта направление в пространстве оси его вращения.

```
                                    Update()
nput.GetKey (KeyCode.A))

nsform.rotation = Quaternion.AngleAxis(30f, new Vector3(1, 2, 3));
```

При использовании кватернионов задается один угол вращения и ось, вокруг которой происходит вращение, поэтому проблемы с не коммутативностью поворотов на разные углы не возникает.

10. Использование ключевых слов “Horizontal”, “Vertical”, “Mouse X”, “Mouse Y” для программирования движения и поворотов 3D-объектов с клавиатуры и мышью.

Конструкция **Input.GetAxis** возвращает значение перемещения объекта по горизонтали и вертикали клавишами клавиатуры и вращения объекта движениями курсора мыши по экрану, связанные соответственно с именными переменными: **Horizontal, Vertical** и **Mouse X, Mouse Y,** настроить чувствительность которых можно в окне, открывающемся командой меню:

Edit/Project Settings/Input

```
                        Update()
 dX = Input.GetAxis ("Horizontal"); //клавиши: A, D (стрелк
                                                        >)
 dZ = Input.GetAxis ("Vertical"); //клавиши: W, S (стрелки:

form.Translate              (dX,            0,            dZ
nslate()  для  перемещения  -  аналог  функции  Rotate()  д
ния
 dXm = Input.GetAxis ("Mouse Y"); //движение курсора по верти
 dYm = Input.GetAxis ("Mouse X"); //движение курсора по гори
form.Rotate              (dXm,            dYm,            0
```

11. Создание и использование шаблонов Prefabs, программное создание экземпляров объектов из шаблона.

Prefab – это контейнер (шаблон) для создания клонированных, то есть идентичных по свойствам и содержанию объектов. Для создания контейнера Prefab необходимо «перетащить» мышью объект из окна Hierarchy в область Assets в нижней части окна среды разработки. При этом в шаблоне сохраняются все присвоенные исходному

объекту компоненты и установленные для них свойства, включая поведение объекта на основе добавленного ему программного кода в компоненте Script.

Для создания клонированных объектов на сцене в редакторе среды Unity необходимо перетянуть из области Assets созданный Prefab на сцену и затем повторить операцию столько раз, сколько потребуется клонированных объектов.

Для генерации клонированных объектов программным кодом из области Assets используется метод Instantiate()

Метод Instantiate(a,b,c) имеет 3 аргумента:

a – имя переменной для связи с шаблоном Prefabs объекта;

b –позиция объекта на сцене;

c – поворот объекта относительно осей координат на сцене.

Генерация объектов методом Instantiate():

```
    GameObject      prefub;      //публичная      переменная      pref
                                                                Update


                                          (Input.GetKeyDown(KeyCode.Space


Vector3 position = new Vector3(0,0,0);
//            позиция            в            центре            цен
Instantiate(prefub1, position, Quaternion.identity);}}
```

12. Создание и использование триггеров, назначение функций OnTriggerEnter(), OnTriggerExit(), OnTriggerStay().

Триггеры – обычно невидимые на сцене объекты, коллайдеры которых не взаимодействуют с физикой и не имеют твердой оболочки. Они используются для управления другими объектами при попадании движущегося объекта в область их коллайдера. Скрипт можно добавлять на один из объектов – на триггер или движущийся объект. Но лучше на триггер! И анализировать имя объекта, который будет входить в триггер

Триггеры обрабатываются тремя функциями:

1) void OnTriggerEnter(Collider col)

{If (col.name==”Cube”) {обработка входа}}

Вызывается при входе в диапазон триггера. Вызывается один раз

2) void OnTriggerExit (Collider col)

{If (col.name==”Cube”) {обработка выхода}}

Вызывается при выходе из диапазона триггера. Вызывается один раз

3) void OnTriggerStay (Collider col)

{If (col.name==”Cube”) {обработка нахождения в триггере}}

Метод будет непрерывно вызываться, когда он находится в пределах диапазона запуска

Свойство col.gameObject будет ссылаться на объект, который взаимодействует с триггером; col.gameObject.name даст его имя. Видимость триггера отключается путем снятия галочки в Mesh Renderer, а установить свойства триггера – в настройках Collider поставить галочку Is Trigger

13. Настройка и принципы использования 3D-звука, обработка кратких и длительных звуков на сцене в среде Unity.

Для воспроизведения звуков
в **Unity** необходимо задать три компонента:
AudioClip – звуковой файл, **AudioSource** – объект-источник воспроизведения звука, **AudioListener** – объект-прослушивающий звук.

Компонеты **AudioClip** и **AudioSourse** нужно назначить, а компонент **AudioListener** привязан к камере (по умолчанию).

Импорт звука:

1. Создать в окне Assets папку для звуков и перетащить в нее звуковые файлы.

2. Каждый загруженный звук настроить на панели **Inspector:**

Force To Mono – переход от стерео к моно звуку

Load In Background – загрузка в фоновом режиме (фоновый звук)

Preload Audio Data – предварительная загрузка аудиоданных

Load Type – загружаемый тип (для небольших звуков – **Decompress on Load**)

Compression Format – формат сжатия (для музыки – **Vorbis**, для короткого звука – **PCM** или **ADPCM**)

Для непрерывного воспроизведения звука (фоновой мелодии – **2D**-звука) нужно просто добавить компонент источник воспроизведения **AudioSource** к камере и указать для него проигрываемый звуковой файл **AudioClip** в **Assets**. При этом прослушиватель звука **AudioListener** по умолчанию уже привязан к камере. При этом создание программного кода для воспроизведения такого звука не потребуется – он будет проигрываться сразу после запуска сцены.

Для того чтобы звук воспроизводился при возникновении какого либо события на сцене или по команде, необходимо добавить компонент **AudioSource** к игровому объекту, а не к камере. При этом для звука в **Assets** нужно сбросить флажок **Play On Awake** (проиграть с момента активизации), а ползунок **Spatial Blend** установить в положение **3D** – трехмерный звук, громкость которого будет зависеть от расстояния между источником и приемником звука. После этого необходимо создать ***программный код*** для запуска звука при совершении события.

Для запуска звука в Unity можно действовать по следующей схеме. Прежде всего, необходимо:

•добавить компоненту **AudioSourc**е к объекту, который должен воспроизводить звук;

•связать переменную **AudioClip** в этой компоненте с соответствующим звуковым файлом в **Assets**;

•настроить параметры звука в компоненте **AudioSourc**е.

Далее, для звука, который должен звучать ***продолжительно*** (например, звук движения танка) необходимо:

•объявить объектную переменную типа **AudioSource** для источника звука и булевскую переменную, которая хранит состояние запущен ли звук;

• в методе Start() проинициализировать объектную переменную как компоненту **GetComponent<AudioSource>();**

•в условиях запуска/остановки звука использовать для объектной переменной соответственно методы **Play()** и **Stop()**, а также установить соответствующее значение **true/false** для булевской переменной состояния звучания.

```
ource          source_tank;          //          источник          зву
  isPlaying    =    false;    //    переменная    для    включения    зву

           Start()
 ...
   source_tank     =     GetComponent<AudioSource>();
                              Update()

x!=0 || z!=0) && !isPlaying)   // если танк двигается и звук не вк
   {                           isPlaying             =             tru
      source_tank.Play();
 == 0 && z == 0 && isPlaying) // если танк не двиг. и звук   вк
   {                                isPlaying          =          fals
      source_tank.Stop();  }
```

14. Добавление встроенных эффектов Unity для событий и использование короутины для задержки выполнения заданных действий.

Добавление эффекта взрыва

Для придания естественности событию попадания снаряда в цель можно использовать имеющиеся в стандартных **Assets** среды Unity эффектах с частицами, которые хранятся в особых разработанных префабах по адресу **Standard Assets/ParticleSystems/Prefabs,** из которых для генерации эффекта взрыва можно выбрать префаб **Explosion.**

В скрипт для префаба снаряда необходимо добавить публичную объектную переменную, например, **explosion1** и связать ее с префабом **Explosion** в списке **Assets** для скрипта префаба снаряда в **Инспекторе:**

public GameObject explosion1;

Затем добавить в функции **OnCollisionEnter()** в условие проверки попадания в цель строку с генерацией префаба взрыва

Instantiate(explosion1, gameObject.transform);

Короутина для запуска выстрела танка:

IEnumerator botshoot() {

canshoot = false; //указываем, что танк-бот стрелять пока не может

//определяем координату для положения снаряда танка-бота

Vector3 forwardofstvol = stvol.transform.position + stvol.TransformDirection(Vector3.forward*4f);

//создаем снаряд из префаба снаряда в требуемой координате относительно ствола
GameObject newcore = Instantiane(core, forwardofstvol, stvol.rotation);

yield return new WaitForSeconds(3f); //ждем 3 секунды (время «перезарядки»)

canshoot = true; //указываем, что танк может сделать выстрел }

Сопрограммы – **короутины (coroutines)** в Unity выполняются параллельно программе в течение некоторого времени. Этим они отличаются от большинства функций, заставляющих программу ждать окончания своей работы

Короутины для своего выполнения используют встроенную функцию **IEnumerator**. Главным компонентом в теле короутины является ключевое слово **yield**, временно прерывающее ее работу, возвращающее управление основной программе и в следующем кадре возобновляющее сопрограмму с прерванной точки

15. Принципы создания автономно управляемых объектов ботов и обнаружения ботом объектов на сцене методом Raycasting.

БОТ - автономная копия объекта, созданная на основе префаба, которая управляется программно

без участия пользователя.

Алгоритм создания бота на сцене

1. СОЗДАТЬ ПРЕФАБ из уже имеющегося на сцене объекта (танка) с его скриптом.

При этом все экземпляры префаба, размещаемые на сцене (боты – танки противника), будут обладать функциональностью, определяемую заданным в скрипте исходного танка-игрока кодом.

2. ДОБАВИТЬ НА ИСХОДНЫЙ ТАНК-ИГРОКА СФЕРИЧЕСКИЙ ТРИГГЕРНЫЙ

КОЛЛАЙДЕР с радиусом, соответствующим расстоянию, на котором будет обнаруживаться танк-игрока другими танками-противника (ботами) при попадании их в этот коллайдер при

движении по сцене танка-игрока и начнется «оживление» танка-бота с выполнением действий,

предусмотренных созданным для него скриптом.

3. СОЗДАТЬ НОВЫЙ СКРИПТ ДЛЯ ПРЕФАБА – источника экземпляров танков-ботов (танков

противника)

ОПРЕДЕЛИТЬ ПЕРЕМЕННЫЕ

```
                                        System.Collection
                                System.Collections.Generi
                                                UnityEngin
c       class       bot       :       MonoBehaviour
t  movespeed  =  0.25f;  //  скорость  передвижения  танка-бо
t  rotspeedtank  =  0.1f;  //  скорость  поворота  танка-бо
t rotspeedbash = 0.5f; // скорость поворота башни танка-бо
t  speedcore  =  3f;  //  скорость  снаряда  танка-бо
ic    Transform    bash;    //    для    управления    башн
ic    Transform    stvol;    //    для    управления    ствол
ic  GameObject  core;  //  для  ссылки  на  префаб  снаря
 canshoot = true; // для определения, может ли танк-бот произвес
```

```
ел
life = 3; // для определения максимального количества попадан
                                                          танк-б
```

!!! НЕОБХОДИМО ПРОИНИЦИАЛИЗИРОВАТЬ ОБЪЕКТНЫЕ ПЕРЕМЕННЫЕ через Inspector –

указать на соответствующие объекты.

ДОБАВИТЬ В СКРИПТ БОТА МЕТОД OnTriggerStay(Collider other)

В методе определяется, какие действия должен выполнять танк-бот при нахождении в триггере

танка-игрока (танку-игроку присвоен тэг Player).

private void OnTriggerStay(Collider other) {

if (other.tag == "Player") // если бот попал в триггер танка-игрока выполняем действия А, Б, В и Г {

МЕТОД «БРОСАНИЯ ЛУЧЕЙ» для определения момента, повернута ли башня на игрока

Метод Physics.Raycast создает «луч» в заданном направлении.

RaycastHit hit;

//переменная для определения объекта попадания «луча»

If(Physics.Raycast(bash.position,bash.TransformDirection(Vector3.forward), out hit))

//если выпущен луч из башни в направлении относительно нее – вперед

{

If((hit.transform.tag==”Player”) && canshoot)

//если луч попал в коллайдер игрока и можно выстрелить

StartCoroutine(botshoot());

//запускаем короутину для выстрела танка-бота

}

16. Программирование динамических эффектов на html-странице: фиксация заголовка, замена текста на рисунок, увеличение размеров картинки.

```
idth=thisФиксация                                                заголовк
 style="position:fixed;    top:0px;  width:  100%;  z-index:2;  tex
:center; margin:0px; padding:0px" >ТЕКСТ</div>
```

Замена текста на рисунок:

```
>t>
.on                                                              toi
ment.getElementById("t1").innerHTML="<img    id='t1'    onmouseout='totext(
univer.jpg'                                                        />
.on    totext()    {document.getElementById("t1").innerHTML="<span    id='t
seout='toim()'>БГТУ</span>"}
.pt>
 щелчке по слову <span id="t1" onmousedown="toim()">БГТУ</span> оно заменяет
университета</p>
```

…

Увеличение размеров картинки:

```
            src="logo.jpg"              onmouseover="t.width*
seout="this.width=this.width/5" />
```

17. Настройка проекта для обработки щелчков мышью по 3D-объектам сцены.

Для обработки щелчков мышью по 3D-объектам сцены, требуется убедиться что на сцене присутствует объект Event System. Затем нужно добавить камере компонент PhysicsRaycaster, для преобразования кликов по 2d экрану в клики в пространстве. После чего в скрипте на нужном объекте сцены реализовать интерфейс

IPointerClickHandler, чтобы обрабатывать какие действия выполнять при клике на данный объект.

18. Программирование обработки щелчка мышью по кнопке на CANVAS.

Во первых, при программировании щелчка мыши по кнопке, стоит убедится, что на сцене присутствует EventSystem, а на камере есть компонент PhysicsRaycaster. Далее на Canvas создаётся кнопка на основе типа Button. Затем либо на самой кнопке, либо на каком-нибудь из игровых объектов на сцене создаётся скрипт, в котором описывается функция, которая должна будет вызываться при клике на кнопку. В конце в инспекторе кнопки находим событие click, перетаскиваем на него объект к которому привязан созданный ранее скрипт и выбираем в меню нужную функцию, которая должна вызываться при клике.

19. Программирование обработки надвижения и ухода курсора мыши с кнопки на CANVAS.

Убедимся, что на сцене есть EventSystem, если нет, то добавим.На камеру добавим компонент PhysicsRaycaster. На Canvas добавим кнопку типа Button. Создаём на каком-нибудь объекте сцены скрипт с функциями MyButtonHover и MyButtonLeave.

У кнопки в инспекторе добавим компонент Event Trigger

Добавим с помощью Event Trigger 2 события Pointer Enter и Pointer Leave.

В инспекторе добавим на событие Pointer Enter кнопки объект с созданным ранее скриптом и выберем из списка функций функцию MyButtonHover

В инспекторе добавим на событие Pointer Leave кнопки объект с созданным ранее скриптом и выберем из списка функций функцию MyButtonLeave

20. Программирование отрисовки на CANVAS таблицы результатов эксперимента размером 3x4.

Таблица состоит из массива текстовых полей, в поля записываются результаты работы с симулятором. Ячейки заполняются последовательно частично вручную, частично автоматически по формулам. Для работы с таблицей на информационной панели для практики должны быть предусмотрены кнопки Button: кнопка для записи значения в таблицу, кнопка для отображения таблицы и кнопка для очистки.

Также на информационной панели должно быть текстовое поле **InputText** для ввода полученных значений со шкалы прибора и занесения его в табл. В ходе выполнения лабораторной работы измерения записываются в нужные ячейки таблицу по нажатию кноп **«Записать»**, для просмотра содержания таблицы использовать событие наведение курсора на кнопку **«Таблица»**, а для очистки таблицы от записей - наведение курсора на кнопку **«Очистка»**. Вся таблица – заголовки, названия полей и т.п. строится из текстовых объектов UI соответствующего размера с фиксированным текстом, а для ячеек, куда нужно записывать результаты эксперимента, используется символ подчеркивания или минуса, чтобы было проще находить нужные ячейки таблицы при записи в них значений по нажатию кнопки «Записать».

Закрытие таблицы public GameObject Table;

```
public void Close() { Table.SetActive(false);}
```

Открытие таблицы private int clickcounter = 0;

```
c                    void                      ControlTable
       (clickcounter        %        2         ==
{Table.SetActive(true);                       clickcounter++
else
{Table.SetActive(false);                       clickcounter+
}}}
```

Очистка [SerializeField] public Text T1; [SerializeField] public Text T2; //текстовые поля, где записываются значения

```
c                          void                              Clear

T1.text                              =                          "-
T2.text = "-"; и так для всех текстовых полей ...}
```

Все эти методы вешаются через скрипт на таблицу и привязываются к соответствующим кнопкам по событию On Click () в инспекторе. В раздел On Click() перетаскивается таблица и из списка выбирается нужный метод для конкретной кнопки, например Clear() для кнопки очистки

21. Программирование обработки и расчетов данных, занесенных в таблицу результатов эксперимента размером 3x4.

22. Семантический анализ текста в информационных системах, семантический вопрос.

Чтобы обеспечить диалог в обучающей системе необходимо, прежде всего, текст представить в формализованном виде. При этом текст в обучающей среде рассматривается как семантический объект, в котором все ключевые понятия и объекты связаны смысловыми цепочками и может быть представлен в виде текстовой базы знаний и формализован как семантическая сеть.

При работе с текстом как с базой знаний решаются различные интеллектуальные задачи (не имеющие однозначного алгоритма выполнения и требующие предварительного анализа), в частности, задача поиска точных ответов на задаваемые в ходе диалога вопросы.

Вопрос, задаваемый на поиск отношения между ключевыми объектами (понятиями) в базе знаний – это **семантический вопрос**. Семантический (смысловой) вопрос требует выдачи точечного ответа, в отличие от поискового вопроса, ответом на который является список ссылок на различные места в тексте, где просто встречаются искомые в вопросе ключевые объекты.

Алгоритм обработки семантического вопроса сводится к грамматическому разбору вопроса и выделению в нем субъектной, объектной части, действия и вопросного слова. При этом первым этапом обработки семантического вопроса является разбор текста вопроса с формированием соответствующих частей.

23. Структура записей базы знаний на основе html-текста информационной системы.

Ключевым элементом разработки компьютерных обучающих систем (КОС) и систем автоматического консультирования (САК) является генерация БАЗЫ ЗНАНИЙ на основе имеющихся инф. материалов, которая и будет использоваться для обеспечения диалога с пользователем по содержанию предметной области. База знаний КОС должна содержать все ключевые понятия-объекты и их смысловые связи в едином учебном материале данной предметной области. Структура выглядит так: ПОДЛЕЖАЩЕЕ – СКАЗУЕМОЕ – ДОПОЛНИТЕЛЬНЫЕ ЧЛЕНЫ ПРЕДЛОЖЕНИЯ

Наиболее компактной формой хранения СС является список дуг. Ключевым отличием списка дуг СС от списка дуг обычного графа является наличие типов отношений (связей) с направлением, обозначаемых собственно дугами, а СС является направленным графом.

В сложных текстах информационных систем могут встречаться обороты в скобках, сложносочиненные предложения и предложения с однородными членами. Для таких предложений в исходном тексте необходима соответствующая обработка. В тексте следует выполнить замену местоимений на названия объектов. Затем осуществляется удаление из текста оборотов, не несущих смысловой нагрузки. После выполнения операций выполняется разбиение текста на простые предложения. Следующим этапом работы является разбиение каждого предложения на отдельные семантические блоки (триады), несущие самостоятельную смысловую нагрузку.

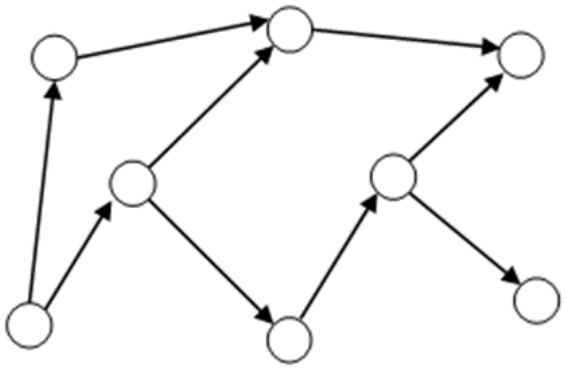
После предварительной обработки исходного текста в соответствии с изложенными выше правилами создается текстовая База знаний для обучающей или информационной системы, состоящая из соответствующего списка триад: ПОДЛЕЖАЩЕЕ – СКАЗУЕМОЕ – ДОПОЛНИТЕЛЬНЫЕ ЧЛЕНЫ ПРЕДЛОЖЕНИЯ

Такую базу знаний как список триад можно хранить в виде двумерного массива на языке JavaScript

```
var knowledge = [ [ "Физика",  "- это", "наука об окружающем мире и ….." ], ];
```

Поскольку ответ в результате диалога выводится на HTML-страницу, то в записях Базы знаний допускается применение HTML-тегов, в т. ч. тегов <img> для рисунков, формул и различных схем.

24. Представление базы знаний в виде семантической сети, объекты, субъекты и действия.



В общем случае СС представляет собой направленный граф с вершинами (узлами) и дугами, которые их связывают.

Под СС для базы знаний понимают граф, в вершинах которого находятся информационные единицы - объекты (сущности), а дуги характеризуют отношения - связи (действия) между ними, которые интерпретируют эти сущности в заданной предметной области.

СС связывает объекты, субъекты и действия. Основным элементом сети является действие, которое передает отношение между субъектом и объектом. Действие связывает субъект (носитель действия) и объект (на кого/что направлено действие или посредством чего оно реализуется).

25. Принцип членения предложений в тексте на триады для записей в базу знаний.

В лингвистике для анализа семантики (смыслового содержания) текста разработан принцип разбиения текста на триады:

- исходную часть сообщения - ТЕМУ,
- новую часть сообщения – РЕМУ,
- связующий член, выражаемый глагольным СКАЗУЕМЫМ.

При этом все предложения в тексте разбиваются на две информационные единицы - тему и рему, и выражаемый глагольным сказуемым тип отношения между ними.

В предложениях текстового учебного материала обучающей или информационной системы необходимо выделять субъект, объект и действие.

Субъект – это то лицо или предмет, которое производит действие.

Объект – это то лицо или предмет, на которое направлено действие или посредством чего оно выполняется.

26. Предварительная обработка исходного текста для подготовки записей в базу знаний.

Одной из задач – организация **диалога**(обмена текстовыми сообщениями между **обучающей средой** и **обучаемым** в процессе обучения)

Чтобы обеспечить диалог в обучающей системе необходимо:

1) представить текст в формализованном виде
2) все ключевые понятия и объекты должны быть связаны смысловыми цепочками
3) использовать алгоритм обработки семантического вопроса(выделить в нем **субъектную(объектную) части**, **действия** и **вопросного слова**)

Для текста:

1) заменить местоимения соответствующими названиями объектов(~~*Он*~~ *Электродный потенциал может*)
2) опускать не несущие прямого смысла обороты и не поддающиеся анализу предложения(~~*К счастью, Таким образом и т.д.*~~)
3) разбить СПП, предложения с однородными сказуемыми и предложения с оборотами в скобках на несколько простых.
4) убрать текст в скобках

27. Алгоритм генерации семантической сети из сложных текстов информационной системы.

АЛГОРИТМ ГЕНЕРАЦИИ СС ДЛЯ СЛОЖНЫХ ТЕКСТОВ:

1. В анализируемом тексте следует выполнить замену местоимений на соответствующие названия объектов.
2. Затем осуществляется удаление из текста оборотов, не несущих смысловой нагрузки.
3. После выполнения данных операций выполняется разбиение текста на простые предложения.

4. Следующим этапом работы является разбиение каждого предложения на отдельные семантические блоки (триады), несущие самостоятельную смысловую нагрузку.

28. Правила формирования записей базы знаний в текстовый массив на языке JavaScript. (в душе не ебу то, не то)

В каждой строке двумерного текстового массива Базы знаний, состоящей из трех элементов триады, хранится:

· в первом столбце блок существительного – подлежащее (субъект с атрибутами),

· во втором столбце блок глагола – сказуемое (действие с атрибутами),

· в третьем столбце блок существительного – дополнительные члены предложения (объект с атрибутами).

//КОД КАК ПРИМЕР

Пример записи массива триад в виде двумерного массива на языке JavaScript:

```
var knowledge =

[

//текстовые триады

[ "измерения ЭДС концентрационных гальванических элементов",

"являются",

"очень удобным, надёжным и точным методом экспериментального определения различных свойств веществ и растворов электролитов" ],

[ "схемы материального баланса в электродных пространствах",

"можно составить",

"пользуясь числами переноса ионов для любых концентрационных элементов с переносом" ],

[ "электрохимия",

"- это",
```

"раздел химической науки, в котором рассматриваются системы и межфазные границы при протекании через них электрического тока, исследуются процессы в проводниках, на электродах (из металлов или полупроводников, включая графит) и в ионных проводниках (электролитах)."],

//триады с картинкой

["правильно разомкнутые гальванические цепи, эквивалентные между собой",

"выглядят",

"следующим образом:
<img src='pictures/ris21_23.jpg'>"],

["схема измерения ЭДС ГЭ компенсационным методом",

"показана",

"на рисунке:
"],

- как выглядят интегральные и дифференциальные кривые титрования щелочью кислот

["интегральные (а, в) и дифференциальные (б, г) кривые титрования щелочью сильной кислоты (а, б) и смеси сильной и слабой кислот (в, г)",

"показаны",

"на рисунке:
"],

//триады с анимацией

["первооткрывателем мира микробов",

"считается",

"Антони ван Левенгук (Antony van Leeuwenhoek, 1632-1723), который жил в Голландии и занимался, в основном, пошивом одежды:
<embed src='1a.swf'/>"]

];

29. Структура диалогового вопроса для получения точечного ответа из базы знаний

Для диалога с БАЗОЙ ЗНАНИЙ с точки зрения семантики предложения будем использовать вопросы, имеющие самую простую структуру:

[вопросное слово] [сказуемое] [подлежащее]

При семантической обработке вопроса **вопросное слово** будет отбрасываться, а в обработке вопроса будут участвовать только его сказуемое и подлежащее.

Ключевым моментом при поиске ответа на вопрос в базе знаний – поиске подходящей строки-записи в текстовом массиве (подходящей триады) будет являться поиск в БАЗЕ ЗНАНИЙ **сказуемого,** соответствующего **сказуемому** в диалоговом вопросе.

На следующем этапе в отобранных по совпадению сказуемых в вопросе и записях-триадах в БАЗЕ ЗНАНИЙ производится отбор по совпадению **подлежащего** в вопросе с **подлежащим** в записях-триадах.

Такой **двойной отбор** в итоге приводит к получению в диалоге **точечного ответа** на заданный вопрос, т.е. выборке из БАЗЫ ЗНАНИЙ только одной-единственной триады, из которой и строится **точный ответ на вопрос** в виде отдельного предложения.

30. Методика отбора записи из семантической сети базы знаний для получения ответа по заданному вопросу.

Алгоритм обработки семантического вопроса сводится к грамматическому разбору вопроса и выделению в нем субъектной, объектной части, действия и вопросного слова.

Вопросное слово в вопросе при поиске ответа в Базе знаний будет отбрасываться.

Ключевым моментом при поиске ответа на вопрос – будет являться поиск в БАЗЕ ЗНАНИЙ сказуемого, соответствующего сказуемому в диалоговом вопросе.

На следующем этапе в отобранных по совпадению сказуемых производится отбор по совпадению подлежащего в вопросе с подлежащим в записях-триадах, что в итоге приводит к получению в диалоге точечного ответа на заданный вопрос.

В вопросе подлежащее и сказуемое, для которых необходимо найти соответствие в базе знаний, могут находится в неподходящих падежах и склонениях, в которых они хранятся в базе знаний.

Поэтому соответствие сказуемых и подлежащих в вопросе и Базе знаний определяется не напрямую, а по некому шаблону, т. н. регулярному выражению.

После получения регулярных выражений для сказуемого и подлежащего они используются в цикле перебора записей базы знаний для проверки их соответствия вопросу.

При помощи регулярного выражения-сказуемого проверяются ячейки второго столбца двумерного массива базы знаний, а при помощи регулярного выражения-подлежащего – последовательно ячейки первого и третьего столбцов базы знаний, т. к. русский язык допускает перестановку слов.

На третьем этапе, если совпадений по подлежащему и сказуемому вместе не найдено, то выполняется поиск только по подлежащему, чтобы учесть возможность использования в вопросе сказуемого, синонимичного записанному в базе знаний.

Однако в этом случае фактически будет выполняться уже поисковый запрос ключевого понятия в базе знаний и, как правило, ответ на такой вопрос будет не точечным, а состоять из совокупности предложений, соответствующих тем записям в базе знаний, в которых встречается подлежащее из вопроса, но с разными сказуемыми, которые, в свою очередь, могут быть и не синонимичными сказуемому в вопросе.

31. Использование шаблонов из регулярных выражений для сказуемого и подлежащего в вопросе.

Для поиска **сказуемого** в БАЗЕ ЗНАНИЙ с учетом возможности использования различных форм сказуемого в вопросе и в исходном тексте, хранящемся в БАЗЕ ЗНАНИЙ, его нужно преобразовать в **регулярное выражение** путем замены найденного ранее псевдоокончания на набор псевдоокончаний, встречающихся во всех допустимых формах, для чего используется экземпляр класса **RegExp()**.

```
//создание регулярного выражения для поиска по сказуемому из вопроса
var predicate = new RegExp(words[i]);
```

В предложениях, не являющихся вводными, для глаголов, как правило, используются формы третьего лица. Так, окончания глаголов I спряжения -ет, -ут, -ют будут заменены на принятое в регулярных выражениях перечисление возможных вариантов строки «(ет|ут|ют)».

//для кратких прилагательных нужно захватить следующее за найденным //слово

if (endings[ending][0] == endings[ending][1]){

predicate = new RegExp(words[i] + " " + words[i + 1]);}

i++;}

Когда сказуемое в вопросе найдено, слова, стоящие за сказуемым и образующие подлежащее с относящимися к нему дополнительными членами, также превращаются в регулярное выражение, что позволяет не повторять дословно термины, используемые в исходном тексте.

//создание регулярного выражения для поиска по **подлежащему** из вопроса

var subject_string = words.slice(i + 1).join(".*");

При формировании регулярного выражения в подлежащем выполняется замена пробелов на принятое в регулярных выражениях обозначение произвольной последовательности символов **«.*»**.

Данное обозначение еще добавляется в начале и в конце регулярного выражения. В результате для вопроса «Как рассчитывается площадь квадрата» будет сформировано регулярное выражение **«.*площадь.*квадрата.*»**, которое будет использовано при поиске ответа в базе знаний.

//только если в подлежащем больше трех символов

if (subject_string.length>3)

{var subject = new RegExp(".*" +subject_string +".*");

32. Классы, функции и методы JavaScript для работы с регулярными выражениями в режиме диалога.

Для проверки соответствия текстового выражения в вопросе с текстовыми выражениями, хранящимися в двумерном текстовом массиве БАЗЫ ЗНАНИЙ, и получения в результате требуемого точечного ответа будут использоваться т. н. **регулярные выражения** – объекты, описывающие символьный шаблон.

Регулярные выражения используются для поиска и обработки подстрок в тексте, основанный на использовании метасимволов.

Для разработки программных модулей диалога с БАЗОЙ ЗНАНИЙ будем использовать следующие классы, методы и функции языка программирования JavaScript для работы с регулярными выражениями:

КЛАССЫ:

Класс **RegExp** представляет регулярные выражения - объекты, описывающие символьный шаблон.

Класс **String** определяет методы, использующие регулярные выражения для выполнения поиска по шаблону и операций поиска в тексте с заменой.

ФУНКЦИИ

split – возвращает массив элементов, полученных из исходной строки (если разделитель – пустая строка, т.е. пробел, то возвращается одномерный массив из всех символов строки).

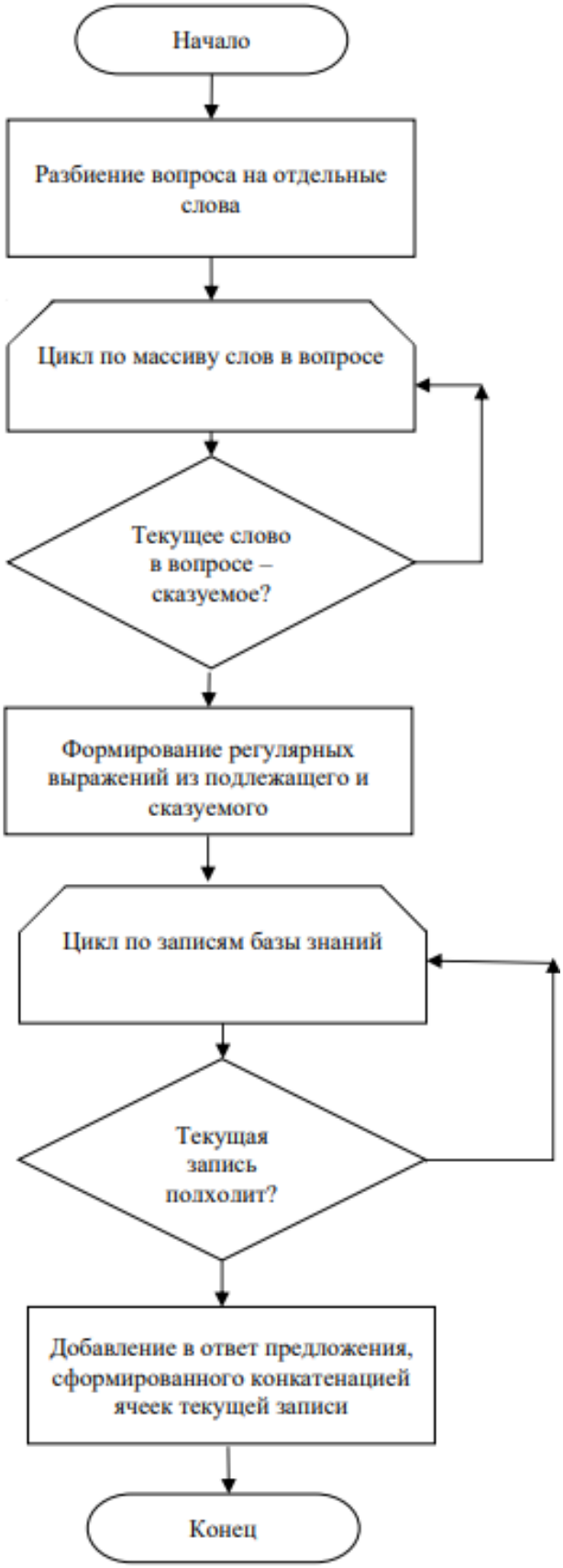
substring – возвращает подстроку исходной строки, начальный и конечный индексы которого указываются параметрами (если параметр один, отбрасывает все символы до указанного индекса).

slice - возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами, за исключением последнего символа.

МЕТОД:

test – метод, используемый для проверки, соответствует ли строка проверяемому регулярному выражению.

33. Общая блок-схема алгоритма получения точечного ответа на вопрос к базе знаний.



В вопросе подлежащее и сказуемое, для которых необходимо найти соответствие в базе знаний, могут находится в неподходящих падежах и склонениях, в которых они хранятся в базе знаний. Поэтому соответствие сказуемых и подлежащих в вопросе и Базе знаний определяется не напрямую, как их точное соответствие, а по некоему шаблону, т. н. регулярному выражению, которые строятся на основе сказуемого и подлежащего в вопросе. После получения регулярных выражений для сказуемого и подлежащего они используются в цикле перебора записей базы знаний для проверки их соответствия вопросу. При помощи регулярного выражения-сказуемого проверяются ячейки второго столбца двумерного массива базы знаний, а при помощи регулярного выражения-подлежащего – последовательно ячейки первого и третьего столбцов базы знаний, т. к. русский язык допускает перестановку слов.

На третьем этапе, если совпадений по подлежащему и сказуемому вместе не найдено, то выполняется поиск только по подлежащему, чтобы учесть возможность использования в вопросе сказуемого, синонимичного (совпадающему по смыслу) записанному в базе знаний. Однако в этом случае фактически будет выполняться уже поисковый запрос ключевого понятия в базе знаний и, как правило, ответ на такой вопрос будет не точечным, а состоять из совокупности предложений, соответствующих тем записям в базе знаний, в которых встречается подлежащее из вопроса, но с разными сказуемыми, которые, в свою очередь, могут быть и не синонимичными сказуемому в вопросе

34. Структура массива псевдоокончаний, используемого для поиска сказуемых в вопросе.

Соответствие сказуемых в БАЗЕ ЗНАНИЙ и сказуемых, стоящих в середине вопроса, будем определять по совпадению в них наборов последних букв – псевдоокончаний – поскольку в роли сказуемых чаще всего выступают определенные части речи: глаголы и краткие причастия/прилагательные.

Для обработки часто встречающегося в диалоге вопроса:

где используется «псевдосказуемое» такое, в БАЗЕ ЗНАНИЙ должен соответствовать используемый в русском языке оборот речи «псевдосказуемое» – это

Таким образом, в текстовый массив для хранения псевдоокончаний `endings` необходимо добавить и следующую строку:

```
var endings =
        [
…
          ["такое"," – это"] ,
          …
    ]
```

Таблица псевдоокончаний хранится в виде двумерного массива, состоящего из 2-ух столбцов. Массив состоит из 1 и 2 сопряжений, возвратных и кратких прилагательных.

Пример массива псевдоокончаний:

```
var endings =
[
        ["ет","(ет|ут|ют)"],
        ["ут","(ет|ут|ют)"],
        ["ют","(ет|ут|ют)"], //1 спряжение
        ["ит","(ит|ат|ят)"],
["ат","(ит|ат|ят)"],
        ["ят","(ит|ат|ят)"], //2 спряжение
        ["ется","(ет|ут|ют)ся"],
        ["утся","(ет|ут|ют)ся"],
```

```
        ["ются","(ет|ут|ют)ся"], //1 спряжение, возвратные
        ["ится","(ит|ат|ят)ся"],
        ["атся","(ит|ат|ят)ся"],
        ["ятся","(ит|ат|ят)ся"], //2 спряжение, возвратные
        ["ен","ен"],
["ена","ена"],
        ["ено","ено"],
        ["ены","ены"],
        ["ан","ан"],
        ["ана","ана"],
        ["ано","ано"],
        ["аны","аны"],
        ["жен","жен"],
        ["жна","жна"],
        ["жно","жно"],
        ["жны","жны"] //краткие прилагательные
];
```

35. Содержание массива «черный список» для исключения подлежащих при поиске сказуемого.

В используемой методике обработки вопросов на основе псевдоокончаний сказуемых приходится учитывать слова в БАЗЕ ЗНАНИЙ, которые не являются сказуемыми, но имеют совпадающие со сказуемыми окончания и которые будут распознаваться при анализе как сказуемые по ошибке. Для исключения такой ситуации необходимо создать отдельный одномерный текстовый массив со словами в базе знаний с совпадающими окончаниями и использовать его при анализе содержимого базы знаний для исключения таких слов из рассмотрения при обработке вопросов в режиме диалога.

Например, при обработке псевдоокончаний сказуемых: -на, -ны, -ут, -ен, -ет в такой текстовый массив должны быть включены слова, которые имеют те же окончания и встречаются в обрабатываемом текстовом массиве базы знаний (список элементов массива должен быть заполнен по всему текстовому массиву БАЗЫ ЗНАНИЙ, созданному по исходному тексту):

let blacklist = ["замена", "замены", "атрибут", "маршрут", "член", "нет"];

36. Структура функции определения сказуемых в вопросе по псевдоокончаниям.

Функция getEnding()определения сказуемых в вопросе

по псевдоокончаниям

Для анализа сказуемого на совпадения его окончания с

соответствующим ему псевдоокончанием в массиве псевдоокончаний

endings[] определим вспомогательную функцию getEnding(word),

в которой вначале сразу же проверяется, не находится ли

рассматриваемое слово word в списке исключений массива

blacklist[] и поэтому его нужно исключить из дальнейшего

рассмотрения:

```
function getEnding(word)
{
//проверка слова на совпадение по черному списку в массиве blacklist

if (blacklist.indexOf(word)!==-1) return -1;
```

Затем в цикле для всех записей в первом столбце массива endings
производится проверка, не имеет ли это слово одно из псевдоокончаний,
характерных для сказуемого:

```
//перебор псевдоокончаний в массиве endings

for (var j = 0; j < endings.length; j++)
{
```

```
//проверка, оканчивается ли слово word на j-ое псевдоокончание

if(word.substring(word.length-endings[j][0].length)==endings[j][0])

//возврат номера найденного псевдоокончания для сказуемого
return j;
}

//если совпадений нет, то возврат -1

return -1;
}
```

37. Алгоритм работы функции получения ответа на вопрос к базе знаний.

*Параметром **question** для этой функции является текст вопроса, который вводится в текстовое окно вопроса диалогового окна разработанного в пользовательском интерфейсе на Web-странице сайта информационной системы.*

1. *В начале функции **getAnswer()** задаются переменные для фиксации успеха в поиске ответа на вопрос к БЗ и записи ответа на вопрос, который должен быть получен в результате выполнения функции:*

```
//флаг, найден ли ответ на вопрос

var result = false;

//формируемый функцией ответ на вопрос – вначале пустой

var answer="";
```

2. *Затем текст вопроса в параметре **question** готовится к обработке путем уменьшения первой буквы до прописной и отделения знаков препинания от слов вставкой между ними пробелов:*

```
//преобразование текста из параметра функции question функцией small1()

//чтобы сделать первую букву в тексте вопроса прописной

var txt = small1(question);
```

//знаки препинания

var separators = "'\",.!?()[]\\/";

//добавление пробелов перед знаками препинания

for (var i = 0; i < separators.length; i++)

txt = txt.replace(separators[i], " " + separators[i]);

3. *После этого текст вопроса методом* **split()** *разбивается на отдельные слова, ориентируясь на все пробелы в тексте вопроса для его дальнейшего анализа, которые сохраняются в массиве* **words,** *автоматически формируемом из текста методом* **split()**.

   //массив слов и знаков препинания, отделенных пробелами

   var words = txt.split(' ');

   Поскольку вопрос состоит из вопросного слова, сказуемого и подлежащего, то сказуемое оказывается в центре вопроса, а если сказуемое в вопросе будет найдено, то искомое подлежащее будет следовать за ним.

4. *Теперь будем решать задачу поиска сказуемого в вопросе, для чего выполним цикл, перебирающем все слова в предложении вопроса, записанные в массив* **words**. *При этом текущее слово считается сказуемым, если обладает характерным для сказуемых* ***псевдоокончанием.***

   //перебор слов в массиве слов из вопроса

   for (var i = 0; i < words.length; i++){

   //поиск номера псевдоокончания с использованием вспомогательной функции **getEnding()** запись его в переменную **ending**

   var ending = getEnding(words[i]);

   Если псевдоокончание будет найдено, а это эквивалентно возвращаемому значению функции ***getEnding()*** *в виде номера в массиве, отличного от -1, то это сказуемое в вопросе, а подлежащее в вопросе будет следовать сразу после него:*

   if (ending >= 0){

//замена псевдоокончания на набор возможных окончаний, хранящихся //во втором столбце массива

words[i] =

words[i].substring(0, words[i].length - endings[ending][0].length)

+ endings[ending][1];

Таким образом, найденное в массиве ***words[i]*** *сказуемое в вопросе заменяется выражением* ***words[i],*** *но уже с набором соответствующих ему псевдоокончаний из второго столбца массива* ***endings****.*

Для поиска сказуемого в БАЗЕ ЗНАНИЙ с учетом возможности использования различных форм сказуемого в вопросе и в исходном тексте, хранящемся в БАЗЕ ЗНАНИЙ, его нужно преобразовать в регулярное выражение путем замены найденного ранее псевдоокончания на набор псевдоокончаний, встречающихся во всех допустимых формах, для чего используется экземпляр класса ***RegExp()****.*

//создание регулярного выражения для поиска по сказуемому из вопроса

var predicate = new RegExp(words[i]);

В предложениях, не являющихся вводными, для глаголов, как правило, используются формы третьего лица. Так, окончания глаголов 1-го спряжения -ет, -ут, -ют будут заменены на принятое в регулярных выражениях перечисление возможных вариантов строки «(ет|ут|ют)».

//для кратких прилагательных нужно захватить следующее за найденным //слово

if (endings[ending][0] == endings[ending][1]){

predicate = new RegExp(words[i] + " " + words[i + 1]);}

i++;}

Когда сказуемое в вопросе найдено, слова, стоящие за сказуемым и образующие ***подлежащее*** *с относящимися к нему дополнительными членами, также превращаются в регулярное выражение, что позволяет не повторять дословно термины, используемые в исходном тексте.*

//создание регулярного выражения для поиска по подлежащему из вопроса

var subject_string = words.slice(i + 1).join(".*");

При формировании регулярного выражения в подлежащем выполняется замена пробелов на принятое в регулярных выражениях обозначение произвольной последовательности символов «.».*

*Данное обозначение еще добавляется в начале и в конце регулярного выражения. В результате для вопроса «Как рассчитывается площадь квадрата» будет сформировано регулярное выражение «.*площадь.*квадрата.*», которое будет использовано при поиске ответа в базе знаний.*

Будем считать в дальнейшем, что выражение для подлежащего в вопросе должно содержать больше трех символов:

//только если в подлежащем больше трех символов

if (subject_string.length>3){

var subject = new RegExp(".*" +subject_string +".*");

Полученные регулярные выражения используются при проходе по всем записям массива в базе знаний.

*Ячейки **второго столбца** проверяются на соответствие регулярному выражению **predicate**, полученному из **сказуемого**.*

*Регулярное выражение **subject**, полученное из **подлежащего**, используется для проверки ячеек как **первого**, так и **третьего** столбцов в двумерном массиве базы знаний, поскольку предложения в вопросе и исходном тексте могут быть сформированы с противоположными по смыслу сказуемыми (“состоять из” <–> “входить в состав”).*

//поиск совпадений с шаблонами среди связей семантической сети

for (var j = 0; j < knowledge.length; j++){

if (predicate.test(knowledge[j][1]) &&

(subject.test(knowledge[j][0]) || subject.test(knowledge[j][2])))

{

//создание простого предложения из семантической связи

answer+=big1(knowledge[j][0] + " " +

knowledge[j][1] + " " + knowledge[j][2] + ". ");

result = true;}}

Поскольку в вопросе может быть использовано сказуемое, синонимичное используемому в тексте, предусмотрен повторный проход по строкам базы знаний без проверки сказуемого.

//если совпадений с двумя шаблонами нет,

if (result == false){

//поиск совпадений только с шаблоном подлежащего

for (var j = 0; j < knowledge.length; j++)

{

if ((subject.test(knowledge[j][0]) ||

subject.test(knowledge[j][2])))

{

//создание простого предложения из семантической связи

answer+=big1(knowledge[j][0] + " " + knowledge[j][1] + " " + knowledge[j][2] + ". ");

result = true;

}}}}}

//если ответа нет

**if(!result)answer = "Ответ не найден.
";}**

38. Процедура подготовки и разбиения текста вопроса на отдельные слова в функции получения ответа.

Текст вопроса подготавливается путем

путем уменьшения первой буквы до прописной **small1(question);**,

и отделения знаков препинания от слов вставкой между ними пробелов

```
var separators = "'\",.!?()[]\\/";
for (var i = 0; i < separators.length; i++)
txt = txt.replace(separators[i], " " + separators[i]);
```

текст вопроса методом split() разбивается на отдельные слова, которые сохраняются в массиве words

39. Процедура поиска сказуемого в вопросе в функции получения ответа.

Вопрос разбивается на отдельные слова, для этого перед всеми знака и препинания добавляется пробел функцией replace -> replace(symbol[i],” “+symbol[i]); После чего методом split(“ “) разбивается на массив слов.

Далее каждое слово определённым образом обрабатывается пока не будет найдено сказуемое, а именно:

Слово проверяется на наличие в blacklist, чтобы исключить ситуацию, когда за сказуемое будет принято слово из темы или ремы.

Слово проверяется по массиву псевдоокончаний, если слово не содержит соответствующего псевдоокончания, то поиск переходит к следующему слову, если же псевдоокончание найдена, то оно заменяется на соответствующий паттерн, для регулярного выражения, на основе которого будет производится сравнение со столбцом действия в триадах базы знаний.

40. Процедура формирования регулярного выражения для сказуемого в функции получения ответа.

Для поиска сказуемого в БАЗЕ ЗНАНИЙ с учетом возможности использования различных форм сказуемого в вопросе и в исходном тексте, хранящемся в БАЗЕ ЗНАНИЙ, его нужно преобразовать в регулярное выражение путем замены найденного ранее псевдоокончания на набор псевдоокончаний, встречающихся во всех допустимых формах, для чего используется экземпляр класса ***RegExp()***.

//создание регулярного выражения для поиска по сказуемому из вопроса

```
var predicate = new RegExp(words[i]);
```

В предложениях, не являющихся вводными, для глаголов, как правило, используются формы третьего лица. Так, окончания глаголов 1-го спряжения -ет, -ут, -ют будут заменены на принятое в регулярных выражениях перечисление возможных вариантов строки «(ет|ут|ют)».

//для кратких прилагательных нужно захватить следующее за найденным //слово

```
if (endings[ending][0] == endings[ending][1])

{

predicate = new RegExp(words[i] + " " + words[i + 1]);}

i++;

}
```

41. Процедура формирования регулярного выражения для подлежащего в функции получения ответа.

После того, как в тексте было найдено сказуемое, всё что будет идти после него будем считать подлежащим, однако поскольку оно может незначительно отличаться от подлежащего в базе знаний, то на основе этого набора слов составляется регулярное выражение вида: .*word_1.*word_2.*word_3.* - где word_1, word_2, word_3 - слова которые идут после сказуемого, а конструкция ‘.*’ - означает, что между словами может быть любая последовательность символов. Затем на основе этой строки создаётся объект регулярного выражения, и с помощью метода test() сравнивается на соответствие с 1 или 3 частью триад базы знаний.

42. Процедура формирования ответа в функции получения ответа на основе метода test().

МЕТОД:

test – метод, используемый для ***проверки***, соответствует ли строка проверяемому регулярному выражению.

Регулярное выражение **subject**, полученное из **подлежащего**, используется для проверки ячеек как **первого**, так и **третьего** столбцов в двумерном массиве базы знаний, поскольку предложения в вопросе и исходном тексте могут быть сформированы с противоположными по смыслу сказуемыми (“состоять из” <–> “входить в состав”).

//поиск совпадений с шаблонами среди связей семантической сети

```
for (var j = 0; j < knowledge.length; j++)
{
if (predicate.test(knowledge[j][1]) &&
(subject.test(knowledge[j][0]) || subject.test(knowledge[j][2])))
{
//создание простого предложения из семантической связи
answer+=big1(knowledge[j][0] + " " +
knowledge[j][1] + " " + knowledge[j][2] + ". ");
result = true;
}
}
```

Поскольку в вопросе может быть использовано сказуемое, синонимичное используемому в тексте, предусмотрен повторный проход по строкам базы знаний без проверки сказуемого.

```
//если совпадений с двумя шаблонами нет,
if (result == false){
//поиск совпадений только с шаблоном подлежащего
for (var j = 0; j < knowledge.length; j++)
{
if ((subject.test(knowledge[j][0]) ||
  subject.test(knowledge[j][2])))
{
//создание простого предложения из семантической связи
answer+=big1(knowledge[j][0] + " " + knowledge[j][1] + " " + knowledge[j][2] + ". ");
result = true;
```

}}}}}}

//если ответа нет

**if(!result)answer = "Ответ не найден.
";**

}

43. Назначение и использование библиотеки jQuery для организации интерфейса диалога с базой знаний.

Диалоговый пользовательский интерфейс с Базой знаний на Web-страницах информационной системы представляет собой использование интерактивных всплывающих диалоговых окон по запросам пользователя на Web-страницах. Для создания интерактивных сайтов чаще всего используется библиотека JavaScript **jQuery**.

Библиотека **jQuery** реализует архитектуру на основе концепций, положенных в основу языка разметки **HTML** и языка каскадных таблиц стилей **CSS,** со следующими базовыми возможностями:

– может обращаться к любому элементу объектной модели Web-документа **DOM** и предлагает механизм селекторов;

– может работать с событиями;

– осуществляет различные визуальные эффекты с **изменением внешнего вида страницы с использованием CSS;**

– имеет различные JavaScript-**плагины**, предназначенные для создания элементов пользовательских интерфейсов, например, диалоговых окон.

В **HTML**-разметку диалоговой Web-страницы добавить следующие элементы:

строку с тегом <script> для подключения файла библиотеки jQuery **`jquery.js;`**

```
<script src="jquery.js"></script>
```

ФУНКЦИЯ ПЛАВНОГО ПОКАЗА-СКРЫТИЯ ДИАЛОГОВОГО ОКНА

```
on toggleDialog(){
 if(dialogOn){        //анимация закрытия диалогового окна
("#dialog").animate({"margin-left":"-25px"}, 1000, function() {});
 dialogOn=false;
 }
 else{          //анимация открытия диалогового окна
("#dialog").animate({"margin-left":"-300px"}, 1000, function() {});
 dialogOn=true;
 clearInterval(timer);
 }}
```

44. Структура сайта и диалоговой Web-страницы с интерфейсом на основе диалоговых окон.

Для реализации на Web-странице диалога с Базой знаний на основе интерфейса с диалоговыми окнами необходимо:

1. В **папку сайта** добавить файлы:
   - библиотеку jQuery – файл **`jquery.js`**,
   - файл разработанного кода на языке JS для обработки диалога
   - файл внешней таблицы разработанных для диалога CSS-стилей
2. В **HTML**-разметку диалоговой Web-страницы добавить следующие элементы:
   - строку с тегом <script> для подключения файла библиотеки jQuery
   - строку с тегом <script> для подключения кода для обработки диалога
   - строку с тегом <link> для подключения внешней таблицы CSS-стилей
   - в теге <body> загрузить оператором **`onLoad()`** созданную дополнительно функцию организации интерфейса для диалогового окна для Web-страницы, например, с именем **`dialog_window()`**.

Общая структура Web-страницы с диалогом может выглядеть следующим образом:

```
         lang="ru"          xmlns="http://www.w3.org/1999/xhtml
>
le>Диалог          с          базой          знаний</titl
a                    charset="utf-8"
ipt                                  src="jquery.js"></scrip
ipt                                  src="dialog.js"></scrip
```

```
k     href="style.css"     type="text/css"     rel="stylesheet"
d>
                                          onLoad="dialog_window()

y>
l>
```

45. Структура функции и использование в ней CSS-стилей формирования общего интерфейса диалогового окна.

Вводим переменную, используемую для активизации диалога

```
var dialogOn = false;
```

ОПРЕДЕЛЯЕМ ФУНКЦИЮ ФОРМИРОВАНИЯ ДИАЛОГОВОГО ОКНА:

```
function dialog_window(){
```

Добавляем на страницу диалоговое окно как новый раздел *div* с идентификатором

***dialog*, задаем ему CSS-стиль *dialog* и надвигаем его левое поле на страницу на 25px:**

```
document.body.innerHTML+=

"<div id='dialog' class='dialog' style='margin-left:-25px;'>"+
```

Здесь используется внешний CSS-стиль:

```
.dialog{position:fixed;top:50%;left:100%;width:300px;height:430px;margin-
left:-25px;  margin-top:-215px;padding:10px;border:#000  2px  solid;border-
radius:10px;background:#fff;color:#000;z-index:1500;font-size:12pt;}
```

Добавляем в раздел диалогового окна новый раздел со строкой *«Нажми, чтобы спросить!»*, задаем ему CSS-стиль *label* и команду на выполнение функции выдвижения диалогового окна *toggleDialog()* щелчком мыши:

```
"<div class='label' onclick='toggleDialog()'>Нажми, чтобы спросить!</div>"+
```

Здесь используется внешний CSS-стиль:

```
.dialog  .label{transform:rotate(270deg);width:430px;height:20px;text-
align:center;overflow:hidden;display:inline-block;margin-left:-71%;margin-
top:68%;cursor:pointer;}
```

Добавляем также в раздел диалогового окна новый раздел со строкой *«История»* и задаем ему CSS-стиль *header*

" История: "+

Здесь используется внешний CSS-стиль:

.dialog .header{ text-align:center;margin-top:-200px;}

Добавляем в раздел диалогового окна еще один новый раздел, задаем ему ему CSS-стиль *history* и задаем ему идентификатор *history*:

" "+

Здесь используется внешний CSS-стиль:

dialog .history{height: 300px;overflow-x:hidden;overflow-y:scroll;}

Добавляем в диалоговое окно последний новый раздел со CSS-стилем оформления *question* в виде поля формы с начальным текстом,исчезающим при щелчке мышью и идентификатором *Qdialog,* а также кнопкой *«Спросить»* с командой для запуска функции *ask()*

**" <input id='Qdialog' placeholder='Введите вопрос'/>
"+**

"<button onclick='ask(\"Qdialog\")'>Спросить</button> "+

Здесь используется внешний CSS-стиль:

.dialog .history .question{margin-left:40px;background:#99f;}

Наконец, закрываем общий блок для диалогового окна *div*

" "; хуета

46. Структура функции и использование в ней CSS-стилей формирования интерфейса обработки вопроса в диалоговом окне.