1. Принципы дискретизации и квантования для оцифровки мультимедийной информации.
2. Общая структура форматов сжатия мультимедийной информации: JPEG для графики и MPEG для видео
3. Внедрение звука в анимационный клип, типы синхронизации звука в среде Adobe Animate.
4. Создание и редактирование объектов в среде 3ds MAX. Использование модификаторов.
5. Использование освещения, материалов и текстур в среде 3ds MAX.
6. Методы редактирования поверхностей 3D-объектов в среде 3ds MAX
7. Назначение и принципы использования компонент Physics, Collider и Materials в среде Unity.
8. Организация движения, вращения и масштабирования объектов в среде Unity на языке C#. Использование методов Rotate() и Translate().
9. Понятие о кватернионах, использование кватернионов для программирования поворотов 3D-объектов.
10. Использование ключевых слов “Horizontal”, “Vertical”, “Mouse X”, “Mouse Y” для программирования движения и поворотов 3D-объектов с клавиатуры и мышью.
11. Создание и использование шаблонов Prefabs, программное создание экземпляров объектов из шаблона.
12. Создание и использование триггеров, назначение функций OnTriggerEnter(), OnTriggerExit(), OnTriggerStay().
13. Настройка и принципы использования 3D-звука, обработка кратких и длительных звуков на сцене в среде Unity.
14. Добавление встроенных эффектов Unity для событий и использование короутины для задержки выполнения заданных действий.
15. Принципы создания автономно управляемых объектов ботов и обнаружения ботом объектов на сцене методом Raycasting.
16. Программирование динамич. эффектов на html-странице: фиксация заголовка, замена текста на рисунок, увеличение размеров картинки.
17. Настройка проекта для обработки щелчков мышью по 3D-объектам сцены.
18. Программирование обработки щелчка мышью по кнопке на CANVAS.
19. Программирование обработки надвижения и ухода курсора мыши с кнопки на CANVAS.
20. Программирование отрисовки на CANVAS таблицы результатов эксперимента размером 3x4.
21. Программирование обработки и расчетов данных, занесенных в таблицу результатов эксперимента размером 3x4.
22. Семантический анализ текста в информационных системах, семантический вопрос.
23. Структура записей базы знаний на основе html-текста информационной системы.

24. Представление базы знаний в виде семантической сети, объекты, субъекты и действия.
25. Принцип членения предложений в тексте на триады для записей в базу знаний.
26. Предварительная обработка исходного текста для подготовки записей в базу знаний.
27. Алгоритм генерации семантической сети из сложных текстов информационной системы.
28. Правила формирования записей базы знаний в текстовый массив на языке JavaScript.
29. Структура диалогового вопроса для получения точечного ответа из базы знаний
30. Методика отбора записи из семантической сети базы знаний для получения ответа по заданному вопросу.
31. Использование шаблонов из регулярных выражений для сказуемого и подлежащего в вопросе.
32. Классы, функции и методы JavaScript для работы с регулярными выражениями в режиме диалога.
33. Общая блок-схема алгоритма получения точечного ответа на вопрос к базе знаний.
34. Структура массива псевдоокончаний, используемого для поиска сказуемых в вопросе.
35. Содержание массива «черный список» для исключения подлежащих при поиске сказуемого.
36. Структура функции определения сказуемых в вопросе по псевдоокончаниям.
37. Алгоритм работы функции получения ответа на вопрос к базе знаний.
38. Процедура подготовки и разбиения текста вопроса на отдельные слова в функции получения ответа.
39. Процедура поиска сказуемого в вопросе в функции получения ответа.
40. Процедура формирования регулярного выражения для сказуемого в функции получения ответа.
41. Процедура формирования регулярного выражения для подлежащего в функции получения ответа.
42. Процедура формирования ответа в функции получения ответа на основе метода test().
43. Назначение и использование библиотеки jQuery для организации интерфейса диалога с базой знаний.
44. Структура сайта и диалоговой Web-страницы с интерфейсом на основе диалоговых окон.
45. Структура функции и использование в ней CSS-стилей формирования общего интерфейса диалогового окна.
46. Структура функции и использование в ней CSS-стилей формирования интерфейса обработки вопроса в диалоговом окне.

1. Принципы дискретизации и квантования для оцифровки мультимедийной информации.

Цифровой звук — это аналоговый звуковой сигнал, представленный посредством дискретных численных значений его амплитуды.

Оцифровка звука — процесс преобразования аналогового сигнала в цифровой.

Оцифровка звука включает в себя два процесса:

• процесс дискретизации (осуществление выборки) сигнала по времени • процесс квантования по амплитуде.

Дискретизация по времени:

Процесс дискретизации по времени — процесс получения значений сигнала, который преобразуется, с определенным временным шагом — шагом дискретизации. Количество замеров величины сигнала, осуществляемых в одну секунду, называют частотой дискретизации или частотой выборки. Чем меньше шаг дискретизации, тем выше частота дискретизации и тем более точное представление о сигнале нами будет получено.

Чтобы оцифрованный сигнал содержал информацию о всем диапазоне слышимых частот исходного аналогового сигнала (0 — 20 кГц) необходимо, чтобы выбранное значение частоты дискретизации составляло не менее 40 кГц. **Линейное (однородное) квантование амплитуды:**

Квантование по амплитуде — процесс замены реальных значений амплитуды сигнала значениями, приближенными с некоторой точностью. Каждый из 2N возможных уровней называется уровнем квантования, а расстояние между двумя ближайшими уровнями квантования называется шагом квантования. Если амплитудная шкала разбита на уровни линейно, квантование называют линейным (однородным).

Этот способ оцифровки сигнала — дискретизация сигнала во времени в совокупности с методом однородного квантования — называется импульсно-кодовой модуляцией, ИКМ.

Оцифровка видео - процесс преобразования видеосигнала из внешнего источника в цифровой видеопоток при помощи персонального компьютера и запись его в видеофайл с целью последующей его обработки, хранения или воспроизведения.

Внешним источником могут выступать видеокамеры, магнитофоны, DVD-проигрыватели, потоковое вещание в сети, ТВ-тюнеры, ресиверы цифрового телевидения и другие устройства.

Процесс оцифровки видео включает себя преобразование аналогового видеосигнала в цифровой поток при помощи видеокарты или платы видеозахвата.

В качестве приложений, которые позволяют захватывать видео и записывать его на жесткий диск, используют видеоредактор либо

видеорекордер. Так как несжатое видео имеет большой поток данных для его уменьшения применяют сжатие при помощи видеокодеков.

В качестве контейнера для обмена и передачи видео- и звуковых данных применяются контейнеры AVI, MOV и прочие.

Оцифровка изображений:

Оцифровка изображений возможна с помощью методов векторной и растровой графики. Векторные изображения создаются как совокупности линий, векторов, точек. Растровые же изображения формируются в виде множества точек (пикселей), распределенных по строкам и столбцам.

Любой цвет может быть представлен комбинацией трех основных цветов. В модели RGB за основные три цвета приняты красный, зеленый и синий, которые характеризуются своей яркостью.

При оцифровке изображение с помощью объектива проецируется на светочувствительную матрицу m строк и n столбцов, называемую растром. Каждый элемент матрицы – мельчайшая точка изображения, состоящая из трех светочувствительных (т.е. регистрирующих яркость) датчиков красного, зеленого и желтого цвета. Далее происходит оцифровка яркости каждой точки изображения по каждому цвету последовательно по всем строкам растра.

Если для оцифровки яркости каждой точки использовать по одному байту (8 бит) на каждый из трех цветов (всего 3x8 = 24 бита), то система обеспечит представление 2^{24}= 16,7 млн. распознаваемых цветов.

Таким образом, происходит оцифровка изображений в устройствах, излучающих свет, мониторах, телевизорах.

На бумаге изображение не излучает световых волн и формируется на основе отраженной волны от окрашенных поверхностей, которые должны поглотить (т.е. вычесть) все составляющие цвета, кроме того, который мы видим. Цвет поверхности можно получить красителями, которые поглощают, а не излучают и должны быть дополняющими: голубой (Cyan = B + G), дополняющий красного; пурпурный (Magenta = R + B), дополняющий зеленого; желтый (Yellow = R + G), дополняющий синего. Для повышения контрастности применяется еще черный (blacK). Модель CMYK названа по первым буквам слов Cyan, Magenta, Yellow и последней букве слова blacK.

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

Оцифровка изображений:

Оцифровка изображений возможна с помощью методов векторной и растровой графики. Векторные изображения создаются как совокупности линий, векторов, точек. Растровые же изображения формируются в виде множества точек (пикселей), распределенных по строкам и столбцам.

Любой цвет может быть представлен комбинацией трех основных цветов. В модели RGB за основные три цвета приняты красный, зеленый и синий, которые характеризуются своей яркостью.

При оцифровке изображение с помощью объектива проецируется на светочувствительную матрицу m строк и n столбцов, называемую растром. Каждый элемент матрицы – мельчайшая точка изображения, состоящая из трех светочувствительных (т.е. регистрирующих яркость) датчиков красного, зеленого и желтого цвета. Далее происходит оцифровка яркости каждой точки изображения по каждому цвету последовательно по всем строкам растра.

Если для оцифровки яркости каждой точки использовать по одному байту (8 бит) на каждый из трех цветов (всего 3x8 = 24 бита), то система обеспечит представление 2^{24}= 16,7 млн. распознаваемых цветов.

Таким образом, происходит оцифровка изображений в устройствах, излучающих свет, мониторах, телевизорах.

На бумаге изображение не излучает световых волн и формируется на основе отраженной волны от окрашенных поверхностей, которые должны поглотить (т.е. вычесть) все составляющие цвета, кроме того, который мы видим. Цвет поверхности можно получить красителями, которые поглощают, а не излучают и должны быть дополняющими: голубой (Cyan = B + G), дополняющий красного; пурпурный (Magenta = R + B), дополняющий зеленого; желтый (Yellow = R + G), дополняющий синего. Для повышения контрастности применяется еще черный (blacK). Модель CMYK названа по первым буквам слов Cyan, Magenta, Yellow и последней букве слова blacK.

MPEG 4 – один из самых популярных форматов видео, который недавно начал поддерживаться Формат обеспечивает хорошее изображение и звук, имеет лучшее соотношение размера файла и качества содержимого контента.

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 Iкадрами – 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. Типичная последовательность кадров выглядит следующим образом:

IBBPBBIBBPBBIBB… Соответственно, последовательность кадров в соответствии с их номерами будет воспроизводиться в следующем порядке: 1423765…

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.

Для того, чтобы импортировать в документ Flash звуковой файл, нужно выбрать пункт «Import» или «ImporttoLibrary» в меню «File». Можно также воспользоваться библиотекой общего использования Sounds в подменю CommonLibraries меню Windows.

Для звука создаётся отдельный слой, на него помещается экземпляр нужного образца-звука, создаётся последовательность кадров нужной длины (назовём её звуковой дорожкой).).

Типы синхронизации звука:

Начать - Данный вид синхронизации исключает возможность наложения звуков.

Остановить - Режим Stop останавливает воспроизведение экземпляров того же звука.

Событие - Данный тип синхронизации используется по умолчанию. Он начинает воспроизводиться, когда проигрывается ключевой кадр, в котором он расположен, и исполняется полностью, независимо от длины линейки, даже если мувиклип останавливается. То есть если экземпляр пятисекундного звука поместить в кадрах, которые проигрываются за секунду, то он все равно прозвучит до конца.

Поток - Режим синхронизации Stream используется для синхронизации потокового звука и анимации для воспроизведения в Web. Flash подстраивает мультипликацию под темп потокового звука. Потоковый звук имеет приоритет перед визуальным рядом: если Flash не может прорисовывать кадры мультипликации достаточно быстро, он пропускает их, а целостность звучания сохраняется. В отличие от звуков события, потоковый звук прекращается, если останавливается анимация. Также потоковый звук никогда не может играть дольше, чем воспроизводится количество кадров, которое он занимает

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

Основные действия, производимые с объектами:

- Перемещение (**move**)
- Масштабирование (**scale**)
- Вращение (**rotate**)
- Выравнивание
- Клонирование

Объекты создаются на основе простых примитивов – куб, сфера, тора и т.д.

Создание трехмерных объектов **называется моделированием.** Моделирование производится при помощи **сплайнов.**

При моделировании на основе сплайнов объекты создаются с помощью форм **Shape.**

При моделировании на основе объектов 2 этапа:

1. Создание исходного объекта на основе примитива
2. Модификация объекта

Модификация производится:

1. С помощью параметров на основе модификаторов
2. С помощью полигонального моделирования.

Для редактирования сложных 3д объектов используются методы трехмерного моделирования:

1. Моделирование на основе примитивов
2. Использование модификаторов
3. Создание объектов при помощи булевых операций
4. Сплайновое моделирование
5. Правка редактируемых поверхностей

Сплайновые инструменты:

Line; Circle; Arc (дуга); NGon (многоугольник); Text; Section (сеченние); Rectangle (прямоугольник); Ellipse; Donut (кольцо); Star; Helix (спираль)

На основе сплайновых фигур сожно создавать сложные геометрические трехмерные объекты при помощи модификаторов:

- Lathe (вращение вокруг оси)
- Extrude (выдавливание)
- Bevel (выдавливание со скосом)
- Loft (лофтинг)
- Surface (поверхность)
- Sweep (выгнутость)

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

Правильно подобранное освещение является одним из наиболее существенных факторов обеспечения реализма сцены при ее визуализации. Оно создает контрасты между объектами, делает использованные материалы более яркими и выразительными и позволяет настраивать тени объектов.

- **Omni** (Всенаправленный) — отбрасывает лучи равномерно во всех направлениях от единственного точечного источника подобно лампочке без абажура;
- **Target Spot** (Нацеленный прожектор) и **Free Spot** (Свободный прожектор) —распространяют лучи из точки в определенном направлении коническим потоком и освещают область внутри конуса. Различие этих двух источников заключается в том, что направление световых лучей в первом из них строго определено точкой цели (**Target**), а второй источник такой точки цели не имеет и потому направление световых лучей в нем может меняться при вращении источника;
- **Target Directional** (Нацеленный Прямой) и **Free Directional** (Свободный Прямой) —распространяют лучи из плоскости параллельным потоком в определенном направлении и освещают область внутри прямого или наклонного цилиндров. Данные источники различаются между собой тем, что направление световых лучей в первом из них имеет привязку-цель, а второй направлен свободно (направление отбрасываемых им световых лучей изменяется при вращении источника).

В 3D Studio **Max** под **материалом** понимают набор характеристик, которые в разной степени влияют на отображение поверхности объекта в процессе визуализации сцены; от степени удачности подбора **материала** зависит и естественность финального вида моделируемой сцены.

Основные **типы материалов:**

- **Standard (Стандартный)** — Обычный материал.
- **Architectural (Архитектурный)** — Материал с расширенными настройками.
- **Blend (Смешиваемый)** — Состоит из двух материалов, которые смешиваются друг с другом по определенной маске.
- **Composite (Составной)** — Похоже на Blend, позволяет смешивать до 9 материалов с основным.
- **Double Sided (Двухсторонний)** — Два материала, один - для передней стороны, другой - для задней
- **Ink 'n Paint (Нефотореалистичный)** — очень интересный материал, имитирует 2D, эффект рисованности.
- **Matte/Shadow (Матовое покрытие/Тень)** — Принимает только тени, сам материал прозрачен.

- **Mutti/Sub-Object (Многокомпонентный)** — состоит из двух и более материалов, каждый материал соответствует своему ID и будет отображаться на полигонах с таким же ID.

Что бы применить его к объекту можно:

- **Просто перетащить материал на нужные вам объекты.**
- **Нажать на кнопку "Assign Material to Selection" (применить материал к выделенному)**

Текстура *(иногда её называют* **картой***)* — это растровое изображение, накладываемое на поверхность модели для придания ей цвета, свойств окраски или иллюзии рельефа.

Придать рельефность — можно, добавив больше полигонов, но гораздо быстрее будет воспользоваться **картой высот** *(англ.* ***height map****),* которую также иногда называют картой рельефа. Это чёрно-белая текстура, которая позволяет сделать рельеф реалистичным.

Существует несколько видов карт высот, у каждой свои особенности:

- **Bump map** *(англ. bump — кочка, выпуклость)* создаёт иллюзию рельефа, но не меняет геометрию объекта. Для этого на цветовую текстуру компьютер накладывает небольшие искажения, чтобы создать иллюзию неровностей.
- **Parallax map** *(параллакс — иллюзия движения объекта относительно фона, которая видна движущемуся наблюдателю)* меняет положение отдельных участков текстуры при отрисовке. То есть при отрисовке parallax map меняется положение отдельных пикселей, а не вершин.
- **Displacement map** *(англ. displacement — смещение)* меняет геометрию объекта.

Чтобы применить карту текстуры к объекту необходимо:

- раскрыть закладку Maps Редактора материалов
- выбрать необходимый канал настройки характеристики
- задать нужное значение Amount выбранной настройки
- щелкнуть на кнопке с надписью None справа от наименования характеристики материала,
- выбрать в появившемся диалоговом окне соответствующую карту,
- двойным щелчком активизировать ее в активном окне образца Редактора материалов
- задать необходимые значения параметров
- присвоить созданную текстуру материалу активному объекту

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

By Vertex (режим выделения, работающий на уровнях редактирования **Face**, **Edge**, **Polygon**) –выделяются соответствующие подобъекты, прилегающие к вершине, на которой был произведен щелчок мыши.

Ignore Backfacing – позволяет выделять только те подобъекты, которые видны в текущем видовом окне (для сложных полигональных объектов, имеющих множество вершин)

Ignore Visible Edges –позволяет игнорировать ребра и выделять сразу несколько полигонов, находящихся под углом друг к другу

Hide/Unhide All – позволяет прятать выделение

Каждый трёхмерный объект представляется в виде набора полигонов (или же многоугольников). Каждый многоугольник состоит из точек, соединенных линиями.

В 3дс Макс есть возможность редактировать все возможные сплайны, из которых состоит объект. Для этого нужно преобразовать его в Editable Poly. После выбрать любой из типов примитивов во вкладке Modify, которые необходимо отредактировать, и изменить положение примитивов сдвигами либо добавлением новых с помощью модификаторов.

Модификаторы:

Bend (Изгиб) – создание деформации изгиба трехмерных объектов. Для корректного применения модификатора объект должен иметь достаточное количество разбиений в направлении оси изгиба.

Extrude (Выдавливание) – построение объектов с постоянным сечением по высоте. Примеры использования: шестеренки и звездочки, текст, машиностроительные детали и заготовки для стен домов. Для построения выдавливания необходимо создать объект формы – профиль сечения, по которому будет строиться выдавливание. Этим профилем может быть как разомкнутая, так и замкнутая кривая, состоящая из одного или более сплайнов.

Bevel (Выдавливание со скосом) – аналогичный модификатору Extrude по способу построения объектов, но с большими возможностями редактирования профиля выдавливания. Его следует использовать при создании объектов с постоянным сечением и фаской на краях (например, текста)

Lathe (Вращение вокруг оси) – создание тел методом поворота вокруг своей оси половины профиля сечения объекта. Примеры объектов такого рода: посуда, кувшины и вазы, песочные часы, автомобильные фары, гантели и т. д. Наиболее важными настройками модификатора **Lathe** являются задание оси вращения и установка поверхности кругового вращения. По умолчанию расположение оси начинается с центра создания формы и выравнивается с локальной осью **Y** формы.

Taper (конус, заострять) придает объекту вид конуса, имеет Gizmo и Center гизмо, которые можно выделить для управления смещения конуса

Twist (Скручивание) – создание деформации скручивания. Чаще всего он используется при конструировании витых спиралевидных моделей: веревок, сверл, резьбы, кованых решеток, ювелирных украшений и т. п. Для корректного применения модификатора объект должен иметь достаточное количество разбиений в направлении оси изгиба

Loft позволяет нарисовать объект с помощью двух сплайнов, один из которых выполняет роль направляющей, а второй - сечения. Этим он позволяет рисовать сложные 3D объекты.

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

Компоненты определяют поведение объектов в игре. Они - функциональная часть каждого объекта. По умолчанию у всех игровых объектов есть компонент **Transform**. Transform диктует, где расположен игровой объект, и как он поворачивается и масштабируется.

Чтобы физическое поведение было правдоподобной, объект в игре нужно правильно ускорить и задействовать столкновения, гравитацию и другие силы. Встроенный в Unity физические движки обеспечивают вас компонентами для обработки симуляции физики. В Unity есть два отдельных физических движка, один для 3D физики и один для 2D физики. Существует компонент **Rigidbody** для 3D физики и аналогичный **Rigidbody** 2D для 2D физики. **Rigidbody** - это основной компонент, подключающий физическое поведение для объекта. С прикреплённым **Rigidbody**, объект немедленно начнёт реагировать на гравитацию.

Компоненты коллайдера **Collider** определяют форму объекта для физических столкновений. В юнити есть различные виды **Collider**, которые позволяют регулировать область взаимодействия объекта с окружающей средой **Box, Sphere, Mesh, Capsule, Terrain, Wheel**. Размер коллайдера можно изменять

Материалы **Materials** используются совместно с Mesh Renderers , Particle Systems и другими компонентами рендеринга Unity. Они играют роль в определении того, как отображается объект. **Materials** используются, чтобы объект сцены имел более реалистичный вид, можно наложить различные картинки. Реализовать сами текстуры объекта

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

Базовыми операциями управления размещением и трансформацией объекта на трехмерной сцене являются: **Position** – координаты объекта в глобальной системе координат сцены, **Rotation** – поворот объекта относительно локальной системы координат объекта, **Scale** – масштабирование объекта.

Они собраны в компоненте Transform объекта и представлены в окне Inspector. Для управления **этими операциями из кода используются свойства**: **position** – задает позицию объекта на сцене относительно глобальной системы координат сцены **rotation** – задает угол поворота объекта в его локальной системе координат **localScale** – задает масштаб объекта

Масштабирование – для программного масштабирования используется свойство transform.localScale

```
void Update() { transform.localScale += new Vector3 (0.1f, 0.0f, 0.0f); }
```

Для организации движения по оси X (расстояние измеряется в условных единицах) используем transform.position:

```
void Update() { transform.position += new Vector3 (0.1f; 0.0f; 0.0f); }
```

Для поворота в **используются углы Эйлера в свойстве eulerAngles:**

```
void Update () { transform.eulerAngles = new Vector3 (10f, 30f, 20f); }
```

Для непрерывного вращения можно использовать функцию Rotate ():

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

Перемещать любые предметы в пространстве относительно оси координат можно методом transform.Translate

```
void Update()
  { float dX = Input.GetAxis ("Horizontal");
float dZ = Input.GetAxis ("Vertical");
transform.Translate (dX, 0, dZ); }
```

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

Кватернионы управляют вращением объекта. Это один из способов вращения наряду с углами Эйлера. **Кватернион — это ось, относительно которой будем вращать объект и угол, на который мы будем вращать объект относительно этой оси**. Всего у кватерниона четыре компоненты: X, Y, Z и W. XYZ — та самая ось поворота (нормализуем и каждый компонент умножаем на синус половины угла), W — угол поворота (который задается через косинус половины угла). 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.

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

Quaternion.AngleAxis(w, Vector3.right) создает вращение, которое вращает angle градусов вокруг axis.

```
public class Rotation_Quaternion : MonoBehaviour
{
    public float w = 30f;
    public Quaternion quaternion ;
void Start()
    { quaternion = transform.rotation; }

void Update()
    {
        w += 0.05f;
        Quaternion quaternionX = Quaternion.AngleAxis(w, Vector3.right);
        transform.rotation = quaternion * quaternionX;
    }
}
```

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

```
void Update()
{float dX = Input.GetAxis ("Horizontal"); //клавиши: A, D (стрелки: <, >)
float dZ = Input.GetAxis ("Vertical"); //клавиши: W, S (стрелки: ^, вниз)
transform.Translate (dX, 0, dZ);
//Translate() для перемещения – аналог функции Rotate() для вращения
float dXm = Input.GetAxis ("Mouse Y"); //движение курсора по вертик.
float dYm = Input.GetAxis ("Mouse X"); //движение курсора по гориз.
transform.Rotate (dXm, dYm, 0);}
```

Функция Clamp() из класса Mathf задает пределы изменения угла поворота:

Mathf.Clamp(угол, нижний предел, верхний предел);

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. 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 Instantinate(). 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.**

```
  public GameObject prefub;
 void Update()
 {
   if (Input.GetKeyDown(KeyCode.Space))
    {
       Vector3 position = new Vector3(0,0,0);
        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. Отключается видимость путем снятия галочки Mesh Renderer. Is Trigger ставится в настройках коллайдера. 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

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

Для воспроизведения звуков необходимо задать три компонента: **AudioClip – звуковой файл, AudioSource – объект-источник воспроизведения звука, AudioListener – объект-прослушивающий звук** Компонент AudioListener привязан к камере

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

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

Непрерывный постоянный звук: 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**. Для того чтобы звук воспроизводился при возникновении какого-либо события, надо добавить компонент **AudioSource к игровому объекту.** 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**.

- добавить компоненту **AudioSourc**e к объекту, который должен воспроизводить звук; **перетащить музыку в переменную AudioClip**
- настроить параметры звука в компоненте **AudioSourc**e.
- повесить на объект скрипт и объявить в скрипте переменную типа **AudioSource** для источника звука и бул 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 методы **Play()** и **Stop()**, установить соответствующее значение **true/false** для переменной состояния.

```
AudioSource source_tank; // источник звука bool isPlaying = false;
 void Start()  {  source_tank = GetComponent<AudioSource>();  }  void Update() {
if ((x!=0 || z!=0) && !isPlaying)  // если танк двигается и звук не вкл.
    {  isPlaying = true; source_tank.Play();  } if (x == 0 && z == 0 && isPlaying) // если танк не двиг. и звук  вкл. {  isPlaying = false; source_tank.Stop();  }
```

Для разового запуска краткого звука можно действовать проще, используется метод *PlayOneShot()* со свойством *clip.*

```
gameObject.GetComponent<AudioSource>().
        PlayOneShot(gameObject.GetComponent<AudioSource>().clip);
```

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

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 выбрать префаб Explosion.

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

```
public GameObject explosion1;
```

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

```
  private void OnCollisionEnter(Collision col)
    {if (col.gameObject.tag == "Goal")
       { Instantiate(explosion, gameObject.transform); }}
```

Корутины (Coroutines, сопрограммы) в Unity — способ запускать функции, которые должны работать параллельно в течение некоторого времени.

Корутины представляют собой простые C# итераторы, возвращающие IEnumerator и использующие ключевое слово yield. В Unity корутины регистрируются и выполняются до первого yield с помощью метода StartCoroutine. Сопрограмма похожа на функцию, которая может приостановить выполнение и вернуть управление Unity, а затем продолжить работу с того места, на котором остановилась, в следующем кадре

```
IEnumerator YourCoroutine() {
    yield return new WaitForSeconds(0.5f);
    Debug.Log("Прошло пол секунды");
}
StartCoroutine("YourCoroutine");
```

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.

1. **СОЗДАТЬ ПРЕФАБ** из 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 игрока начнется «оживление» танка-бота с выполнением действий, предусмотренных его скриптом. 3. **СОЗДАТЬ НОВЫЙ СКРИПТ ДЛЯ ПРЕФАБА** бота

4. **ОПРЕДЕЛИТЬ ПЕРЕМЕННЫЕ и инициализировать через инспектор**

```
public class bot : MonoBehaviour { float movespeed = 0.25f;
    // скорость передвижения бота float rotspeedtank =
0.1f; // скорость поворота бота float rotspeedbash = 0.5f;  //
скорость поворота башни бота float speedcore = 3f;  //
скорость снаряда бота public Transform bash;  //  для
управления башней public Transform stvol; //  управления
стволом public GameObject core; //  ссылка  на  префаб
снаряда  bool  canshoot  =  true;  //  может  ли  танк-бот
произвести выстрелить int life = 3;                    //
максимальное кол. попаданий
```

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 тег Player).**

```
private void OnTriggerStay(Collider other) {
  if (other.tag == "Player")   {} // если бот попал в триггер танка-игрока выполняем действия }
```

6. МЕТОД «БРОСАНИЯ ЛУЧЕЙ» нужен для определения момента, повернута ли башня на игрока. **Метод Physics.Raycast создает «луч» в заданном направлении.** RaycastHit h; //переменная для определения объекта «попадания» луча

```
If (Physics.Raycast(bash.position.TransformDirection(Vector3.forward), out hit))
If (hit.reansform.tag == “Player”) && canshoot) //если луч попал в коллайдер игрока - можно выстрелить StartCoroutine(botshoot()); //запускаем короутину для выстрела
```

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

```
<!DOCTYPE html>
<html>
  <head>
    <style>#changecolorparagraph:hover { color: rgb(159, 92, 185); }
   </style><script>
function to_img() //заменяет текст на картинку
          {document.getElementById("changetoimgparagraph").innerHTML =
"<img width='30%' height='30%' src='img.jpg'></img>";
          }
function to_text() //возвращает текст
          {document.getElementById("changetoimgparagraph").innerHTML = "<p
id='changetoimgparagraph' onmousedown='to_img()'
onmouseout='to_text()'>Рисунок</p>";
          }</script></head>
<body>
<p>Этот <span style="color:blue" onmouseover="this.style.color='red' "
onmouseout="this.style.color='blue'">синий текст </span>при наведении
мыши становится красным</p>
```

ФИКСАЦИЯ ЗАГОЛОВКА:

```
<div style="position:fixed;  top:0px; width: 100%; z-index:2; background-color:
#FFFFFF; font-size: 24px; text-align:center; margin:0px; padding:0px" >
<p> Web-дизайн и разработка мультимедийных изданий</p></div>
```

УВЕЛИЧЕНИЕ КАРТИНКИ:

```
          <p id="changecolorparagraph">Текст подсвечивается при
наведении</p>
 <img onmouseover="this.width=this.width*1.5"
onmouseout="this.width=this.width/1.5" width="30%" height="30%"
src="img.jpg"></img>
```

ЗАМЕНА ТЕКСТА НА РИСУНОК:

```
<p id="changetoimgparagraph" onmousedown="to_img()"
onmouseout="to_text()">Рисунок</p>
  </body>
</html>
```

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

Для обработки события «щелчок мышью по 3D объекту на сцене»:

1. Добавить в иерархию объектов новый не отображаемый на сцене объект Create/UI/EventSystem

2. В сценарий для объекта необходимо добавить класс using UnityEngine.EventSystems

3. В базовый класс добавить новый интерфейс системы событий IPointerClickHandler

```
using UnityEngine;
using System.Collections;
using UnityEngine.EventSystems;
public class Script11 : MonoBehaviour, IPointerClickHandler {
void Start () {}
void Update () {}    }
```

4.К камере необходимо предварительно добавить компоненту Physics RayCaster для согласования щелчков мыши по 2D-экрану со щелчками по 3D-объектам на сцене

5. На объекте, по которому мы кликаем, должен быть коллайдер, редактируем его размеры Edit Collider

6. Реализуем логику, которая происходит после щелчка, в скрипте – создаем метод public void OnPointerClick(PointerEventData pointerEventData) {…}

18. Программирование обработки щелчка мышью по кнопке на CANVAS.

1) В сцене на основе CANVAS создается рабочий UI-объект и кнопка Button.

2) Затем к объекту добавляется сценарий, который будет вызываться при совершении событий, которые возможны для кнопок.

3) Добавить директивы using UnityEngine.UI и using UnityEngine.EventSystems 4) Выбрать событие для кнопки и создать связь с UI-объектом, к которому присоединен соответствующий сценарий.

5) Выбрать для кнопки в качестве функции имя сценария для добавленного UI-объекта, и в раскрывающемся списке выбрать ту функцию, которая должна выполнить действие, когда будет взаимодействие с кнопкой.

По умолчанию у кнопки событие - OnClick() – щелчок мыши, для других событий необходимо к кнопке добавить компоненту Event Trigger и затем выбрать из списка нужное событие, например OnPointerEnter() – надвижение курсора мыши.

public void OnPointerEnter(PointerEventData eventData) – метод для обработки наведения курсора на кнопку

public void OnPointerExit(PointerEventData eventData) – метод для обработки ухода курсора с кнопки

OnClick() – метод для обработки щелчка

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

public void OnPointerEnter(PointerEventData eventData) – метод для обработки наведения курсора на кнопку

public void OnPointerExit(PointerEventData eventData) – метод для обработки ухода курсора с кнопки

1) В сцене на основе CANVAS создается рабочий UI-объект (например, панель) и кнопка Button

2) Затем к панели добавляется скрипт, который будет вызываться при совершении событий, которые возможны для кнопок. Там можно определить методы закрытия и открытия панели при помощи SetActive(true) или SetActive(false)

3) Добавить директивы using UnityEngine.UI и using UnityEngine.EventSystems

4) По умолчанию у кнопки событие - OnClick() – щелчок мыши, для других событий необходимо к кнопке добавить компоненту Event Trigger и затем выбрать из списка нужное событие OnPointerEnter или OnPointerExit – наведение и увод мыши.

5) Выбрать событие OnPointerEnter() для кнопки и создать связь с UI-объектом, к которому присоединен соответствующий сценарий. Перетащить панель

6) Выбрать для кнопки в качестве функции имя сценария для добавленного UI-объекта, и в раскрывающемся списке выбрать ту функцию, которая должна выполнить действие, когда будет взаимодействие с кнопкой.

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;
public void ControlTable()
{if (clickcounter % 2 == 0)
{Table.SetActive(true); clickcounter++;}
else
{Table.SetActive(false); clickcounter++;
}}}

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

Для этого мы создаем скрипт. В скрипте нужно объявить все текстовые поля, куда будут заноситься данные пользователем или где будут высчитываться значения. Скрипт вешаем на таблицу. Потом для кнопок в событие On Click() в инспекторе перетащим таблицу и выберем нужный метод для нужной кнопки.

```
    [SerializeField]
    public Text T1;
    [SerializeField]
    public Text T2;
    [SerializeField]
    public Text T3;
    [SerializeField]
    public InputField textInput;
    public void WriteToTable()
    {//метод записи данных
        double numericValue;
        if (double.TryParse(textInput.text, out numericValue))
        {
            if (T1.text == "–")
            { T1.text = textInput.text.ToString(); textInput.text = "";}
            else if (T2.text == "–")
            { T2.text = textInput.text.ToString(); textInput.text = "";}}

  private void Update()
    {
        double calculateValue = 0;
        if (T1.text != "–" && T2.text != "–")
        {
            calculateValue = (float)(double.Parse(T1.text) - double.Parse(T2.text));
            T3.text = calculateValue.ToString();
        }}
```

Метод Parse() в качестве параметра принимает строку и возвращает объект текущего типа, например double.Parse(строка) вернет объект типа double из строки – извлечет из строки число. Метод ToString() приводит к строковому представлению

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

Чтобы обеспечить диалог в обучающей системе необходимо 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 как семантическая сеть. 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. В лингвистике для анализа семантики (смыслового содержания) текста разработан принцип разбиения текста на триады:

- исходную часть сообщения – ТЕМУ (субьект),
- новую часть сообщения – РЕМУ (объект),
- 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. ТРИАДА: «ПОДЛЕЖАЩЕЕ»-«СКАЗУЕМОЕ»-«ДОПОЛНИТЕЛЬНЫЕ ЧЛЕНЫ ПРЕДЛОЖЕНИЯ» Такой подход применим только к простым и сложноподчиненным предложениям. В сложных текстах информационных систем могут встречаться обороты в скобках, сложносочиненные предложения и предложения с однородными сказуемыми.

Для таких предложений в исходном тексте случае необходима соответствующая предварительная обработка текста.

В анализируемом тексте следует выполнить замену местоимений на соответствующие названия объектов. Затем осуществляется удаление из текста оборотов, не несущих смысловой нагрузки. После выполнения данных операций выполняется разбиение текста на предложения. Следующим этапом работы является разбиение каждого предложения на отдельные семантические блоки (триады), несущие самостоятельную смысловую нагрузку.

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.

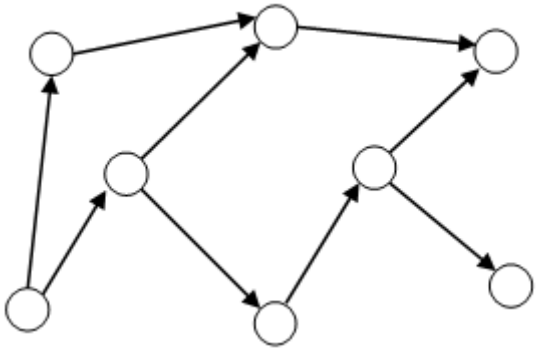
После предварительной обработки исходного текста в соответствии с изложенными выше правилами создается текстовая База знаний для обучающей или информационной системы, состоящая из соответствующего списка триад: ПОДЛЕЖАЩЕЕ – СКАЗУЕМОЕ – ДОПОЛНИТЕЛЬНЫЕ ЧЛЕНЫ ПРЕДЛОЖЕНИЯ

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.

Реализация СС для диалоговой системы и анализатора текста на естественном языке имеет свою специфику – в качестве идентификаторов вершин используются текстовые названия объектов, а не их номера.

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

исходную часть сообщения – ТЕМУ (подлежащее), новую часть сообщения – РЕМУ (дополнительные члены предложения), 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.

Первый блок существительного можно определить как субъектный блок: блок субъекта – того, кто или что производит действие. Второй блок существительного можно определить как объектный блок: блок объекта – того, на кого или на что направлено действие (посредством кого или чего оно реализуется).

Такое представление соответствует схеме в виде триады: субъект – действие – объект

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

Принципы предварительной обработки текста:

1. В анализируемом тексте следует выполнить замену местоимений на соответствующие названия объектов.
2. Затем осуществляется удаление из текста оборотов, не несущих смысловой нагрузки.
3. Разбить сложносочиненные предложения, предложения с однородными сказуемыми и предложения с оборотами в скобках на несколько простых.
4. 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

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

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.

1. **ЗАМЕНИТЬ** местоимения соответствующими названиями объектов
2. **ОПУСКАТЬ** не несущие прямого смысла обороты и не поддающиеся анализу предложения
3. **РАЗБИТЬ** сложносочиненные предложения, предложения с однородными сказуемыми и предложения с оборотами в скобках на несколько простых. Например, убираем текст в скобках. Из текста в скобках создаем два новых предложения.
4. **Предложения должны получиться ЛАКОНИЧНЫМИ и нести только ОДНУ СМЫСЛОВУЮ НАГРУЗКУ**

28. Правила формирования записей базы знаний в текстовый массив на языке JavaScript.

Массив - это упорядоченная коллекция значений. Значения в массиве называются элементами, и каждый элемент характеризуется числовой позицией в массиве, которая называется индексом. Отсчет индексов массивов в языке JavaScript начинается с нуля. Базу знаний как список триад можно хранить в виде двумерного массива на языке JavaScript – у нас получается массив в массиве. **Каждая запись массива knowledge – это отдельный массив из трех строк. Массив не типизирован, на что указывает ключевое слово var, но хранить мы будем строковые константы**

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.

var knowledge = [["Формула вычисления силы трения", "выглядит", "следующим образом: "],];

Каждая триада соответствующем следующему виду:

ПОДЛЕЖАЩЕЕ – СКАЗУЕМОЕ – ДОПОЛНИТЕЛЬНЫЕ ЧЛЕНЫ ПРЕДЛОЖЕНИЯ

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

[вопросное слово] [сказуемое] [подлежащее]

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.

Таблицу псевдоокончаний будем хранить в виде следующего текстового двумерного массива, состоящего из 2-х столбцов:

```
var endings =
 [["ет","(ет|ут|ют)"], //1 спряжение
["ут","(ет|ут|ют)"],
["ют","(ет|ут|ют)"],  ….. и т.д.
```

Для обработки часто встречающегося в диалоге вопроса:

«что такое АААААА?»,где используется «псевдосказуемое» такое, в БАЗЕ ЗНАНИЙ должен соответствовать используемый в русском языке оборот речи «псевдосказуемое» – это, что приводит к ответу типа:«АААААА – это ...»

Таким образом, в текстовый массив для хранения псевдоокончаний endings необходимо добавить и следующую строку:var endings = ["такое"," – это"] ,

Для исключения такой ситуации, когда в базе знаний имеются слова, которые не являются сказуемыми, но имеют совпадающие со сказуемым окончания и которые будут распознаваться при анализе как сказуемые по ошибке, 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):

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

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

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.

Так как типов сказуемых (отношений между объектами) в исходном тексте всегда гораздо меньше, чем типов подлежащих (объектов), то и выбранных записей из Базы знаний, подходящих в соответствии со сказуемым в заданном вопросе будет гораздо меньше.

На следующем этапе в отобранных по совпадению сказуемых в вопросе и записях-триадах из Базы знаний 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.

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

Соответствие сказуемых и подлежащих в вопросе и Базе знаний определяется не напрямую, как их точное соответствие, а по некому 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.

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.

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).

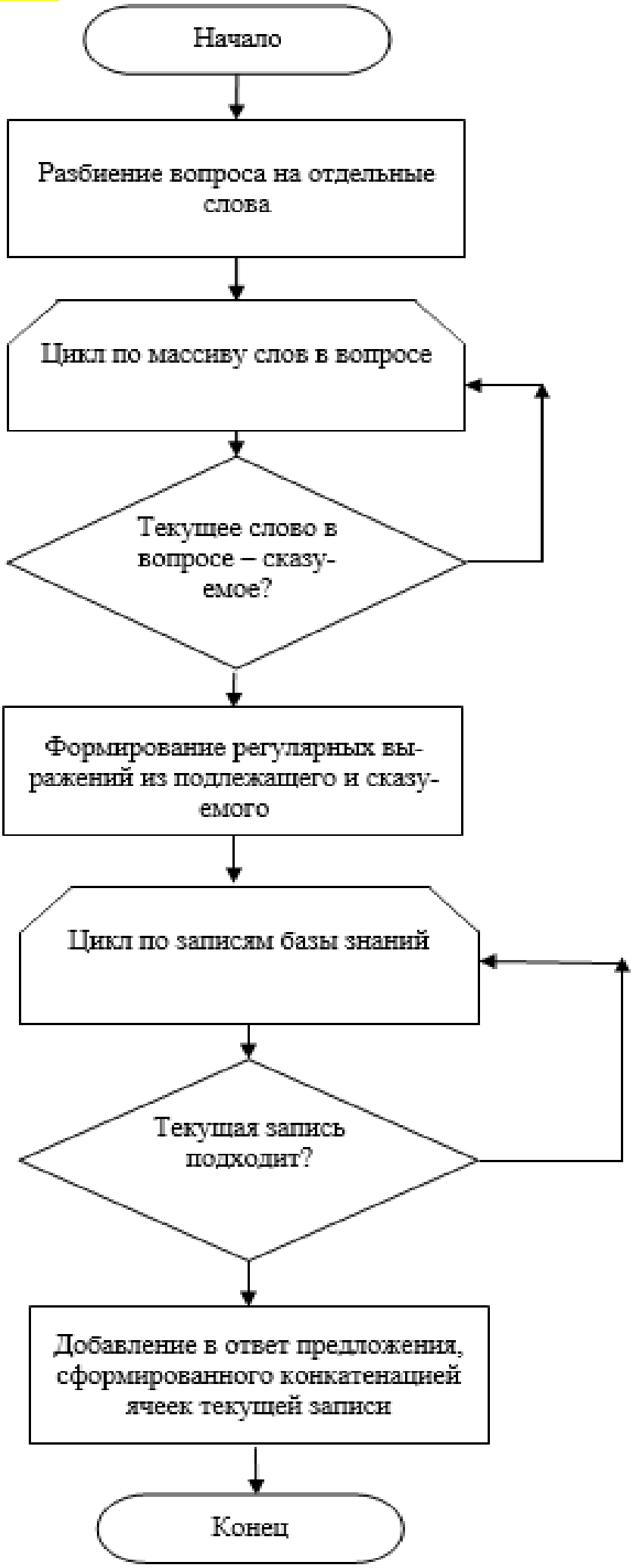
substring – возвращает подстроку исходной строки, начальный и конечный индексы которого указываются параметрами (если параметр один, отбрасывает все символы до указанного индекса).

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.



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

Таблицу псевдоокончаний будем хранить в виде следующего текстового двумерного массива, состоящего из 2-х столбцов:

```
var endings = [
["ет","(ет|ут|ют)"],//1сопряжение
["ут","(ет|ут|ют)"],
["ют","(ет|ут|ют)"], ….. и т.д.
```

Для обработки часто встречающегося в диалоге вопроса:

«что такое ААААА?»

где используется «псевдосказуемое» такое, в БАЗЕ ЗНАНИЙ должен соответствовать используемый в русском языке оборот речи «псевдосказуемое» – это, что приводит к ответу типа:

«ААААА – это ...»

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

```
["такое"," – это"] ,
```

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

Для исключения такой ситуации, когда в базе знаний имеются слова, которые не являются сказуемыми, но имеют совпадающие со сказуемым окончания и которые будут распознаваться при анализе как сказуемые по ошибке, 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):

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

36. Структура функции определения сказуемых в вопросе по псевдоокончаниям.

Для анализа сказуемого на совпадения его окончания с соответствующим ему псевдоокончанием в массиве псевдоокончаний endings[] определяется вспомогательная функция getEnding(word), в которой вначале сразу же проверяется, не находится ли рассматриваемое слово word в списке исключений массива blacklist[]:

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

А 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.

Регулярные выражения, полученные для сказуемого и подлежащего, используются при проходе по 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.

```
//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. <br/>";}
```

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

Для получения ответа на вопрос используется **функция getAnswer(question).** Параметром question для этой функции является текст вопроса, который вводится в текстовое окно вопроса диалогового окна.

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

```
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]);
```

2. 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.

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

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.

```
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.

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.

```
//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.

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 «.*».

```
//только если в подлежащем больше трех символов
if (subject_string.length>3)
{
var subject = new RegExp(".*" +subject_string +".*");
```

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

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

Полученные регулярные выражения используются при проходе по 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.

```
//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. <br/>";}
```

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

Для реализации диалогового окна мы должны выполнить следующие действия:

1. Прежде всего, вы должны добавить библиотеку jQuery в свой проект. Вы можете загрузить последнюю версию с jquery.com:

```
<script src="http://code.jquery.com/jquery-1.9.1.js "></script>
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js "></script>
```

2. Создайте div тег на своей странице и поместите в него содержимое диалогового окна.

```
<div> <p> Здравствуйте, это мой первый диалог с использованием </p>
</div>
```

3. В script тег поместите код jQuery для открытия диалогового окна:

```
<тип сценария="текст / javascript">
$ (документ).готово(функция () {
$(функция () {
 $("#dialog").dialog({

 Автозапуск: false,
модальный: true,
показать: {
 эффект: "слепой",
длительность: 1000
 },
скрыть: {
 эффект: "взрыв",
длительность: 1000 }});});});</script>
```

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

```
<html lang="ru" xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Диалог с базой знаний</title>
 <meta charset="utf-8" />
 <script src="jquery.js"></script>
 <script src="dialog.js"></script>
 <link href="style.css" type="text/css" rel="stylesheet"/>
</head>
<body onLoad="prepare_environment()">
…
</body>
</html>
```

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

```
//переменная, используемая для активизации диалога
var dialogOn = false;
function prepare_environment(){
document.body.innerHTML+=
"<div id='dialog' class='dialog' style='margin-left:-25px;'>"+
"<div class='label' onclick='toggleDialog()'>Нажми, чтобы спросить!</div>" +
"<div class='header'>История:</div>"+
"<div class='history' id='history'></div>"+
"<div class='question'><input id='Qdialog' placeholder='Введите
вопрос'/><br>"+
"<button onclick='ask(\"Qdialog\")'>Спросить</button></div>"+
"</div>";
}
```

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

```
function ask(questionInput){
//переменная для считывания содержания окна ввода вопроса
var question=document.getElementById(questionInput).value;
//активизация диалога
dialogOn=true;
//создание поля для ввода вопроса в новом блоке <div>
var newDiv=document.createElement("div");
newDiv.className='question';
newDiv.innerHTML=question;
//добавление созданного блока в блок диалога
document.getElementById("history").appendChild(newDiv);
//создание поля для вывода ответа в новом блоке <div>
newDiv=document.createElement("div");
newDiv.className='answer';
//получение ответа на вопрос и заполнение им созданного блока ответа
newDiv.innerHTML=getAnswer(question);
//добавление созданного блока в блок диалога
document.getElementById("history").appendChild(newDiv);
//прокрутка истории в самый низ
document.getElementById("history").scrollTop =
                document.getElementById("history").scrollHeight;
//очистка текстового поля для ввода нового вопроса
document.getElementById(questionInput).value="";
}
```