Профильные вопросы.

1. Чем отличается модель памяти С# и С++?
2. Что ты знаешь про массивы?
3. У нас есть 2 массива, на 100 и 1000 элементов, из первого я вытаскиваю 100й элемент, а из 2го 900, эти операции будут разные по времени?
4. Можем ли мы добавлять элементы в массив?
5. Какие структуры данных ты знаешь?
6. Какие бывают типы данных?
7. Чем отличаются ссылочные и значимые типы данных?

CA CD

Fx

Kinematic

Adresables Как работают?

Unitask для чего? Почему не таски? Почему не корутины?

Лист работа

ООП.

1. Что такое абстрактный класс?
2. Делегаты и события что можешь рассказать?

Структуры и классы, отличия, когда что лучше использовать?

Структуры значимый тип и ведут себя как значимый тип, то есть передаются по значению, все значения копируются в новый объект и используется, а классы ссылочный и по факту передаются ссылки на область в памяти, где лежит этот класс.  
Классы умеют в наследование, а структуры нет, но структуры могут реализовывать интерфейсы.

Структуру лучше создавать, когда нам нужно определить тип данных, который состоит из примитивных типов, которые логически должны быть вместе, этих данных не должно быть много, чем больше данных, тем медленнее будет работать структура. К примеру, структура Point, которая будет состоять из X и Y позиции.

String – что за тип, как работать.

String – это ссылочный тип, но ведёт он себя отлично от других ссылочных типов, так как каждый раз создаёт новую строку. Отсюда могут вытекать проблемы с производительностью, так как каждый раз создавать новую строку не выгодно, для работы с ними был придуман тип StringBuilder, он может изменять строку не создавая новую. Его использование имеет смысл при работе с большим количеством строк или их частым обновлением, к примеру в Update, а для не частых обновлений он не так полезен так-как на его постоянное создание тоже выделяются ресурсы, однако имеет смысл закешировать его и использовать. У него, как и у листа есть поле Capacity – скорее всего для того, чтобы не создавать новую строку каждый раз и MaxCapacity

К основным методам StringBuilder можно отнести:

Append() – Присоединение одной строки к другой.

AppendLine() – Присоединение одной строки к другой но следующая строка будет не сразу за первой, а с новой строки.

Remove(startCharIndex, LastCharIndex) – Удаление части строки.

Insert(index, text) – Вставит заданный текст в указанную позицию.

Replace(targetText, newText) – заменит TargetText на NewText.

AppendFormat() – Работает как обычный string.Format().

Clear() – Удалит всё что есть в строке, она станет Empty

ToString() – Преобразует наш Sb в строку.

Типы коллекций в C#

Массивы, листы, словари, хасшсет, арай-лист, очередь, LinkedList.

List.

По факту является массивом в специальной обёртке позволяющей добавлять в него новые элементы, под капотом по факту происходит создание нового массива и копирование в него значений, массив создается не каждый раз, у листа есть поле Capacity, и когда размер листа становится больше Capacity, то в этот момент создается новый лист с новым Capacity. В моих тестах Capacity был больше размера на 4, но при создании пустого листа он был равен 0.

Значимые и ссылочные типы данных.

Стек и куча – и то и другое это область оперативной памяти только. Стек – маленькая до 1мб. Куча – большая, 1.5гб на 32бит и 8 на 64бит. Значимые – стек. Ссылочные – куча.

Есть 3 вида кучи, SOH, LOH, POH,

1 – для маленьких объектов до 85кб,

2 – для больших, более 85кб также в LOH не происходит дефрагментация памяти.

3 – Добавилась совсем не давно с DotNet 5. Специальная куча для закреплённый объектов.

Размер кучи задается при запуске приложения и в отличии от кучи он ограничен лишь физически.

OOP – Основные парадигмы, приведение типов.

Инкапсуляция – Я понимаю это как сокрытие от вмешательства из вне, а всё действие с классом происходит через специальные открытые методы.

Наследование – Концепция создание нового класса потомка на основе родителя, где потомок будет по факту являться родителем.

Полиморфизм – Изменение поведения класса на основе класса родителя, интерфейса или данных, принятых в метод.

Виды полиморфизма:

Ad-hoc полиморфизм – на основе перегрузки методов или конструкторов.

Параметрический полиморфизм – Как я понял на основе дженериков, можем получить любой массив и вернуть оттуда, к примеру первый элемент.

Полиморфизм подтипов – На основе классов и интерфейсов с их перегрузками.

Linq

Что использовал в Linq?

Преоброзование в массивы, листы.

Select – Возвращает список с true или false.

Where – Вернет список с соответствующими условиями.

FirstOrDefault – Найдет первое совпадающие по условиям

Принципы SOLID, DRY, KISS, YAGNI

Принцип SOLID.

S – Принцип единственной ответственности.

Каждый скрипт отвечает за что-то одно, разделение большой сущности на мелкие

O – Принцип открытости – Закрытости кода.

Различные сущности должны быть открыты для расширения, но закрыты для изменения.

Сущности могут изменять свое поведение при этом их исходный код должен оставаться неизменным, а если что-то нужно изменить, то создаем отдельную реализацию с новым поведением.

L – Принцип подстановки Барбары Лисков.

Наследники базового типа должны корректно реализовывать его поведение и не должны противоречить базовому поведению.

I – Принцип разделения интерфейсов.

Интерфейсы не должны быть перегружены, лучше разбить большой интерфейс на несколько маленьких чем иметь нереализованные методы в классе.

D – Инверсия зависимостей. модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций. Вспоминай пример про игру с кораблями или разными платёжными средствами.

DRY – Не дублируй код.

KISS – Делай проще, не стоит специально усложнять задачу, старайся сделать самым простым способом.

YANGNI – Вам это не понадобиться. Грубо говоря, не нужно добавлять то, что вас не просили добавлять.

Паттерны.

По поводу паттернов, что должен понимать сразу, так это:

- Порождающие: отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

- Структурные: отвечают за построение удобных в поддержке иерархий классов.

- Поведенческие: решают задачи эффективного и безопасного взаимодействия между объектами программы.

# Порождающие.

Одиночка.

Создаёт объект в единственном экземпляре тем самым нарушая SRP, но взамен гарантирует наличие единственного экземпляра класса и предоставляет глобальную точку доступа.

Создаём единственный объект, а при попытке создать новый отдаём старый.

# Поведенческие.

Observer.

Создает удобного посредника, который наблюдает за одним классом и в случае достижения нужного состояния запускает нужный код в другом объекте.

Visitor.

Паттерн на основе перегрузки методов в интерфейсе, к примеру у нас есть много оружия и от каждого мы можем получить урон, вместо постоянных If Else с перебором и поиском оружия мы можем создать метод IWeaponVisit куда передать нужное оружие и в зависимости от оружия, дальнобольное или ближнего действия использовать нужную логику, а может мы вообще захотим, чтобы у какого-то юнита была слабость к определённому оружию? Дело за нами!

Comand.

Реализация посредством создания базового класса команды и создание наследников с командой и одним методом – нажатием, при выполнении которого будет происходить разная логика, дальше эти команды можно поместить в словарь, где ключом будут команды, а значением клавиши и когда нажата нужная клавиша то выполняется команда, относящаяся к этой клавише. Это позволяет налету менять значения словаря, а команды останутся теме же.

# Структурные.

Decorator

Создаю интерфейс с методом получения статистик, к примеру как я это делал для замедления от уровня земли он будет возвращать новые статистики.

Его реализую в классе со всеми статистиками, для того чтобы я мог прокинуть этот класс в параметры метода получения статистики.

Создаю абстрактный класс с декоратором, в котором реализую ранее созданый интерфейс, а также закидываю реализацию этого интерфейса в поле + создаю протектед конструктор, который будет получать наш интерфейс и присваивать его в поле и создаю абстрактный метод с получением статистик, который возвращаю в метод от интерфейса.

В самом декораторе я уже могу делать разные расчёты, изменять конструктор как мне захочется и в абстрактном методе вернуть нужное значение.

UnityWebRequest

Переделывал систему загрузки с www на это. Грузил картинки, файлы.

JSON

Работал с системой сохранений на основе Json, парсил конфиги.

# Try cath finali

Попробуй выполнить (1), если не получилось, кэтч (2) при любой ошибке. В финали попадём в любом случае, даже если была ошибка. А также:

1. Если вы используете "return" внутри блока try. \*\*
2. Если вы используете "return" внутри блока catch. \*\*
3. Если вы создаете какое-либо исключение внутри блока catch.
4. Если ваш блок try вызывает исключение, которое не может быть перехвачено вашим блоком catch.

Меш – это?

Меш – это конструкция, используемая графическим конвейером для отрисовки сложных вещей.

Он содержит в себе как минимум коллекцию вершин в 3д пространстве и набор треугольников, которые объединяют эти вершины.

Для отображения 3д модели необходимо 2 компонента, Mesh filter и Mesh renderer.   
Mesh filter – Хранит в себе ссылку на меш

Mesh renderer – Отвечает за его отрисовку.

Как текстура накладывается на объект? Это реализовано с помощью добавления 2д текстурных координат на каждую вершину. Эти координаты называются UV и лежат в диапазоне от 0.0 до 1.1

Boxing Unboxing

Boxing – преобразование значимого типа данных в ссылочный.

Unboxing – Обратный процесс.

Batching/Draw call

Batching – позволяет объединять несколько запросов от процессора к видеокарте в один.

В юнити активируется галкой StaticBatching в инспекторе.

Не на статических включаем галку Enable GPU instancing

Какие алгоритмы ты знаешь?

Поиск:

Линейный – перебор массива по порядку и сравнение с нужным значением.

Двоичный – деление массива пополам, массив должен быть отсортирован.

Сортировка:

Пузырьком – берётся первый индекс и по очереди сравнивается элемент, если элемент больше, чем i++ то поменять их местами.

Вставками – Берётся индекс 1 и копируется, после если всё что больше этого элемента перемещается влево, а сам элемент вставляется на пустое место

Garbage Collector

У него есть 3 поколения объектов, при первом размещении всем присваивается первое поколение. Чем выше поколение, тем более дороже обходится сборка мусора в нем.

Сборщик может сработать в любой момент и в этот момент происходит полная остановка приложения. Который проверяет используется ли объект в программе или нет. Если нет, то его удаляют. Если да, то повышает поколения нужным объектам.  
Чем ниже поколение, тем больше шансов что объект будет удалён и наоборот.

Если объект весит больше 85кб и попадает в LOH кучу, то его поколение сразу равно 2.

Как понимает, что что-то нужно удалить? Между объектами создаются связи на основе графов через которые сборщик понимает востребовательность объектов в программе.

При удалении объекта будет вызван метод Destructor, которые выглядят как название класса без модификатора доступа + ~ перед названием, но по факту это синтаксический сахар, который содержит под капотом метод Finalize, поэтому это одно и тоже.

Когда сборщик мусора собирает объекты он не уничтожает их сразу, а сначала помещает их в специальную очередь, Finalization Queue, исполняет в порядке очереди выполняет методы Finalize и уничтожает объекты.

Про проект с резкой.

По большей части резка происходит так, есть класс земли с полигон2д колайдером состоящим из множества отрезков, а на пиле есть объект для резки, если круг попадает в зону 2д колайдера то вычисляет точки пересечения земли с окружностью пилы, и те что внутри удаляем, а на тех что снаружи создаём новые

Структуры данных.

Односвязный список.

По сути, элемент односвязного списка — это класс, который содержит 2 поля, адрес и класс, адрес указывает на следующий такой-же элемент в памяти.

Если нужно убрать элемент из середины, то нам нужно найти элемент, который был до него и заменить его ссылку на ссылку удаляемого элемента.

Добавление элемента в середину происходит заменой ссылки предыдущего элемента на ссылку нового, а в новом элементе вставляем старую ссылку из прошлого элемента.

Плюсы в быстродействии добавления, а минусы в том, что доступ к элементу O(1) мы можем получить только зная нужный адрес, либо это будет O(n). При добавлении в конец, поскольку для доступа к последнему элементу придётся перебрать все элементы до него, это будет работать либо медленнее, либо так-же как и обычный List

Двусвязный список.

В двусвязном списке, как и в односвязном есть ссылка на следующий элемент, но также есть и ссылка на предыдущий.

Пример двусвязного списка в C# — это LinkedList, имеет под капотом ссылку на последний и первый элемент, а значит может добавлять сразу в начало и в конец списка элементы через AddFirst и AddLast, может добавлять до или после какого-либо элемента, но нужна ссылка на элемент, по которому будем добавлять, AddBefore и AddAfter, ссылку можно получить через Find если хотим искать с начала или FindLast если с конца.

Плюсы по сравнению с односвязным, быстрее доступ к концу и началу, можем выбирать откуда начинать поиск нужного элемента.

Минусы — это больше действий для изменения ссылок на прошлый и следующий элементы.

Делегаты.

Тип в языке C#, который способен хранить ссылку на методы. Для чего нужен? Эвенты принимают ссылки на методы, а по выполнению эвента выполняется нужный код. В Callback, передаём в метод ссылку на другой метод чтобы он был выполнен в нужный момент. Может ссылаться на методы с заранее определёнными возвращаемыми параметрами и с нужными параметрами.

Как оптимизировать RayCast.

Ситуация, когда нам нужно пока палец на экране вести по экрану луч, тут можно вместо создания нового луча в update создать луч один раз, и затем только обновлять его позицию, это гораздо лучше, чем создание луча каждый раз.

Способы оптимизации кода.

Меньше всяких Foreach, Linq и прочего сахара, стандартный For в ряде случаев быстрее.

StringBuilder вместо Sting.

Меньше проверок – больше фпса.

Меньше логов в финальном билде.

Больше переиспользования уже созданных объектов, уменьшит время работы сборщика мусора, не будут тратится ресурсы на создание объектов каждый раз.

Больше атласов для канваса, меньше галок рейкаст таргет и график рейкастов у текстовых компанентов тоже нуда убирать.

# Сколько потоков в Unity?

В unity 2 потока, первый главный, а второй для рендеренга.