**По текстовому заданию:**

* Не совсем понял про стратегия – это поведение, а шаблон – свойства. По сути все поведенческие паттерны это про поведение (состояние тоже про поведение). Тут скорее дело в реализации и подходе. Стратегия – это про подмену поведения извне (т.е. мы можем класть в сущность различные реализации поведения, тем самым как-то будет изменяться поведение сущности – получается композиционный принцип, когда мы можем собирать сущность из разных кусочков, с возможностью замены этих кусочков), а шаблонный метод привязан к иерархии классов и переопределяет некоторые методы/шаги какого-то алгоритма в зависимости от дочернего объекта. Этот вариант менее гибкий, но зачастую бывает удобнее в использовании, чем стратегия.
* Примерчики неплохие, но, советовал бы попытаться немного подробнее расписывать их, чтобы лучше понимать как ты предполагаешь их реализацию, но это на твое усмотрение. В остальном все окей)

**По заданию с пушками:**

* Сразу вижу жесткое несоблюдение код стайла, о котором я говорил на занятии. Публичные поля с нижнего подчеркивание (должно быть с большой буквы). Не везде стоят модификаторы доступы. Местами оставлены комментарии, которые засоряют код (тот же update в AmmoClass – бесполезные комменты надо убирать). Также не везде соблюдаются отступы между методами (когда нет отступов код превращается в сплошной текст – так труднее читать его). Также if, for и тд в одну строку писать нельзя (ammoClass – 23 строка).
* В названиях классов не надо использовать слово Class – это никакой полезной информации не несет. Кроме того я так понимаю этот класс представляет пулю – значит и название лучше сделать bullet или projectile. Также я бы сделал не fire метод а Launch (как бы запуск пули).
* Также не очень, что у тебя публичные поля – вообще забудь про публичные поля. Все поля приватные должны быть, максимум protected, но опять таки с оговорками, которые мы обсуждали, когда говорили про принцип подстановки барбары лисков. Кто угодно может менять публичные поля и ломать класс тем самым – это неправильно. Доступ к полям можно ограничивать логикой методов или свойствами с валидацией данных.
* Также не очень понимаю зачем тебе вообще поле \_ammo, если класс AmmoClass и так может представлять собой пулю (т.е. вешаешь его просто на пулю, потом пушка создает через Instantiate AmmoClass и пуля летит изменяя свой transform просто). Кроме того я бы еще добавил, чтобы пуля уничтожалась через какое-то время, а не только при врезании в коллайдер (у Destroy есть перегрузка, которая принимает еще время, через которое надо уничтожить объект)
* Плохие названия Gun1 Gun2 и тд – лучше давать более осмысленные
* Не понимаю зачем ты создаешь Quaternion’ы и позицию с нулевыми координатами – можно и без этого обойтись (есть Quaternio.Identity + есть еще возможность передать нужную позицию для спавна пули в конструктор например.
* Также очень плохо, что ты используешь для пули GetComponent – ты бы мог просто использовать не GameObject в параметрах, а прям сразу указывать класс AmmoClass, тогда у тебя Instantiate сразу вернет ссылку на нужный компонент и не придется дополнительно использовать getComponent (а в друг вообще такого компонента нет на объекте??)
* Также мультишот лучше было сделать через цикл – без дубляжа кода
* Все магические числа надо выносить в поля, константы и тп (т.е. всех троек, двоек и тп непонятных не должно быть в коде)
* Не очень хорошее решение засовывать проверку кол-ва патронов в выстрел – лучше сделать в таком случае IsCanShoot() и какой-то и контекст будет проверять, можно ли сделать выстрел, в случае если нет, то что – то выводит на экран, а если может, то стреляем.
* Также можно заметить, что много общего кода есть (то же создание пули) – так что тут можно было использовать абстрактный класс вместо интерфейса и вынести какую – то общую логику туда тем самым избежав дубляжа кода
* Плюс можно было интерфейс IGun разделить на два, раз уж пошли через интерфейсы IShootable и IReloadable – и использовать их отдельно, тогда можно было бы IReloadable вообще не использовать для пушки с беск. Патронами
* По поводу переключения пушек – не очень хорошо, что у тебя все в одной сущности. Во первых стратегия должна подменять извне сущности, которая ее использует. Т.е. кто-то должен подменять у игрока пушку (как в примере с шерифом), потому что иначе игрок начинает знать детали реализации всех пушек (из чего они состоят, чем их инитить и тп), а нам этого не надо. А во вторых банально много отственностей – это и обработка ввода и смена пушек и стрельба из них и создание этих пушек
* Вообще, учитывая те знания, которые мы имеем на первом занятии, я бы предложил сделать базовый монобеховский Weapon (возможно общую логику вынести туда, например создание пули) и от него три реализации пушек (тут можно было бы, возможно, потренировать еще и шаблонный метод). Потом сделать WEaponSwitcher, который полностью контролирует логику смены оружия и сделать класс Shooter, который стреляет из оружия, которое ему кладет WeaponSwitcher (по аналогии с шерифом). Пушки можно просто энейблить и дизейблить или разрушать и создавать. И тогда у тебя тут будет четко разбитая логика того кто и за что отвечает и стратегия будет в подмене пушек у Shooter. Также можно сюда добавить ручной бустрапер (как мы делали в примерах, создавая классики Example, чтобы проинитить все). За отлавливание ввода может отвечать отдельная сущность Input (мы тоже посмотрим как ручками делать свой инпут, когда дойдем до зенжекта). А Shooter и WeaponSwitcher будут подписываться на события этого инпута и делать определенные действия. Также, естественно, выделить отдельный класс Bullet для пули и создавать именно ее
* Также прикреплю пару неплохих реализаций для доп изучения от участников курса(но со своими косячками тоже – тут учитывай мои советы:
  + <https://github.com/JimmDiGrizli/UnityPatternCourse/tree/main/Lesson_1/TasksProject/Assets/Task2/Scripts>
  + <https://github.com/SteriksGame/Templates-Project-1-Lesson/tree/main/Assets/Gun%20and%20Trader/Scripts>

**По заданию с трейдером:**

* Очень плохой нейминг классов – уделяй этому больше времени
* Например, я бы назвал самого npc Trader и интерфейс для поведения ITraderBehaviour
* Также не очень опять таки магические числа 1,4 в трейдере, но для примера пойдет
* Также сам трейдер не должен по идее реализовывать интерфейс INPC (это нелогично, т.к. по идее у тебя интерфейс реализуют различные поведения этого трейдера, но из-за нейминга впринципе не очень понятно какой логики ты хотел добиться тут). Сейчас ты можешь случайно вместо поведения положить самого трейдера, то приведет к казусу
* Также не очень тот момент, о котором я говорил ранее – поведение почему-то выбирается самим трейдером. Все что он должен делать – это использовать готовое поведение, которое ему положат, иначе он должен знать про то как создать каждое поведение (это сейчас тут пустые конструкторы и все хорошо, но в реальной игре у каждого поведения будут зависимости и получается трейдер должен будет про всех них знать, что сильно размывает его область ответственности)
* Также логичнее было наверное назвать метод не GetSpeech а Trade
* Также не в коем случае нельзя делать проверку через gameobject.name! Кто то поменяет название в сцене и все посыпится. Я показывал хорошую проверку через TryGetComponent. Сделать надо интерфейс какой-то ITradable например и повесить на игрока, а далее проверять по этому интерфейсу
* Вообще я бы сделал бы какой-то TraderBehaviourSwitcher, который подписан на событие изменения репутации игрока. Например, он бы при смене репутации менял поведение у торговца (при первому уровне менял на фрукты, при втором менял на броню). Таким образом можно, например, заранее менять поведение торговца или даже визуально какие-то вещи менять, еще до того как персонаж подойдет к торговцу. Кроме того логика смены поведения торговца будет обрабатываться отдельной сущностью, а трейдер будет только использовать нужное поведение для торговли, но в целом все сделано неплохо и для проверки примера подойдет – по паттерну понимание есть)
* У самого класса игрока довольно много ответственностей – я бы так же разбил управление и тп, но для примера пойдет
* <https://gitlab.com/boykopb/patterns_course/-/tree/main/patterns_course/Assets/Week_1/HW_3> - вот хорошй довольно пример
* <https://github.com/Olegsander78/GoF_Patterns_Strategy/tree/Trade_Configs/GoF_Strategy/Assets/Task3/Scripts> - и вот еще

Задание с мини игрой:

* Ну тут как-то грустно, что задание вроде с мини игрой какой-то а сущностей совсем мало – уже говорит о том, что попытки как-то разбить это не было
* Опять таки много комментариев – код должен говорить сам за себя
* Опять названия плохие (зачем называть SphereClass, когда можно просто Sphere – смысл не меняется)
* Также события вызывать обязательно через ? chpokEvent?.Invoke, потому что, если никто не подпишется на это событие, то будет ошибка, а знак ? проверку сделает на null и все будет окей
* Также при использовании GetComponent обязательно должен быть аттрибут RequireComponent (я рассказывал об этом в своих видосах), чтобы нужны компонент точно висел на объекте и не было ошибок
* Через OnMouseDown не очень обрабатывать ввод, но до ввода мы еще дойдем. Вообще лучше было сделать сущность отдельную BallClicker какой нибудь. Все таки лопать шарики – это доп ответственность
* Довольно плохое решение использовать int для идентификации – можно было использовать Enum, где перечислить цвета
* Ну Player я думаю тебе тоже понятно, что сильно перегружен и стратегия тут вообще не используется (тут и UI и условия победы по методам и вообще все все все). Поэтому предложу примерный вариант того, как это дело лучше разбить
* Я бы сделал так – абстрактный VictoryCondition у которого событие Completed (т.е. условие выполнено) и от него два наследника, которые отслеживают конкретное условие победы. При создании в них передается список шаров. У шаров есть события того, что по ним кликнули и условия победы подписываются на них (соответственно каждый раз при клике шара надо проверять, условие победы). Также сделал бы класс Level – в который при старте передавал бы созданное и проинициализированное условие победы (в зависимости от нажатой кнопки например – это и было бы стратегией). И в level подисывался на событие выполнения условия и когда оно срабатывает уровень бы уже думал что делать дальше, вывести экран победы или еще что. Также сделал бы какой-то Clicker отдельной сущностью, который кликает по шарам (через те же рейкасты, например) Т.е. тут была бы стратегия в подмене условий, где каждое условие инкапсулирует в себе отслеживание победы и выкидывает ивент об этом наружу (при этом у нас класс уровня не будет разрастаться и даже знать о том какие реализации условий победы есть)
* Класс Level бы тут выполнял роль некоего верхнеуровнего корня игры + избавляемся от большого example в пользу простого бустрапера для инициализации, что тоже хорошо
* В целом тут плохо, что не было попытки разбить игру на части, не говоря уже о применении стратегии, хотя тут она напрашивается с этими условиями победы
* <https://github.com/JimmDiGrizli/UnityPatternCourse/tree/main/Lesson_1/TasksProject/Assets/Task4/Scripts> - в целом вот более менее неплохой пример (но сделан немного не по той схеме, которую я описал), можешь по изучать и учесть мои замечания

**Общее:**

* Обязательно еще раз послушай советы по клин коду с занятия
* И прикреплю несколько ссылок на мои видосы с рекомендациями по инкапсуляции и в целом по коду:
* [**https://www.youtube.com/watch?v=RxqTC\_2l880&t=336s**](https://www.youtube.com/watch?v=RxqTC_2l880&t=336s) **– инкапсуляция**
* [**https://www.youtube.com/watch?v=ZIGJTNrvfjg&t=550s**](https://www.youtube.com/watch?v=ZIGJTNrvfjg&t=550s) **– советы по архитектуре 1**
* [**https://www.youtube.com/watch?v=XHkIc0xesA4&t=208s**](https://www.youtube.com/watch?v=XHkIc0xesA4&t=208s) **– вторая часть**
* [**https://www.youtube.com/watch?v=CCf5COet4Qc**](https://www.youtube.com/watch?v=CCf5COet4Qc) **– третья часть**
* [**https://www.youtube.com/watch?v=jVFXnDd40CE&t=771s**](https://www.youtube.com/watch?v=jVFXnDd40CE&t=771s) **– тут в начале пример про ввод – его полную версию мы разберем на четвертом основном занятии**