**V1**

V1 – самая простая реализация, где при срабатывании триггера на спавн таила, он тут же спавниться в точке спавна.

В этой реализации НЕ учитывается идеальная точка пересечения, а триггер срабатывает, как только коллайдеры пересекаются, из-за этого видно, что таилы пересекаются

--

Движение реализовано через RigidBody, скрипт, который задает вектор направление движения талов (виден в инспекторе) и скрипта который по вектору направление, через заданное кол-во секунд добавляет скорость RigidBody

---

**Дополнительно**

(функционал, который можно убрать, он не влияет не на что)   
для каждого таила по 2 точкам(можно и больше) вычисляться границы таила.

(отдельно) Затем на основе границ каждого таила, вычисляется границы всей карты, состоящей из таилов

(! ВАЖНО!) И т.к. пока нету логики спавна стартовых таилов, первый таил в буфер(список) для расчета границ карты, нужно вносить ручками самому.

**V2**

V2 – В этой реализации находиться идеальная точка пересечения таилов на основе точки пересечения двух векторов (при срабатывании триггера на столкновение, в инспекторе рисуется идеальная точка пресечения и 2 вектора). Используются 2 вектора

1. Это вектор направление, который задаётся отдельным объектом.
2. Это вектор, вдоль которого будет происходить поиск точки пересечения в пространстве (если вектор вдоль оси X (под любым углом), то он найдет точку пересечения на любой координате Y)

В качестве тех точек, которыми должны были бы пересечься коллайдеры (в идеальном мире), используются

1. Это центр коллайдера триггера, который находит метку на таиле, о том, что пора спавнить следующий таил
2. Это центр коллайдера метки на таиле, сообщающий о том, что пора спавнить следующий таил

Все эти расчеты происходят в момент столкновения колайдеров.

После нахождение этой идеальной точки пересечения, т.к. вектор направление движения талов не меняется, то значит у всех таилов он одинаков. А значит **просто вычтя текущую координату центр коллайдера метки на талие из координат идеальной точки**, я получаю вектор, который все так же направлен вдоль направления движения таилов **и так же этот вектор является тем смещением, на которое нужно сдвинут след. таил**, **что бы сохранить расстояние между таилами**

**!!ВАЖНО**. В ЭТОЙ РЕАЛИЗАЦИИ НЕЛЬЗЯ ПО ХОДУ ИГРЫ МЕНЯТЬ ВЕКТОР НАПРОВЛЕНИЯ ВЕКТОРА ДВИЖЕНИЯ ТАИЛА. ОН ДОЛЖЕН ЗАДОВАТЬСЯ ПЕРЕД СТАРТОМ ИГРЫ. (СВЯЗАНО С РЕАЛИЗАЦИЕЙ ОПРЕДЕЛЕНИЯ СМЕЩЕНИЯ ТАИЛОВ)

А ТАКЖЕ В ЭТОЙ РЕАЛИЗАЦИ НУЖНО БЫТЬ ОЧЕНЬ ОСТОРЖНЫМ СО ВСЕМИ TASK, Т.К. ЛЮБАЯ ЗАДЕРЖКАИ ИЗ-ЗА TASK, ОЗНАЧАЕТ, ЧТО НАЙДЕННОЕ СМЕЩЕНИЕ ДЛЯ СЛЕД, ТАИЛА СТАНЕТ НЕ АКТУЛЬНЫМ, Т.К. ТАИЛЫ ПРОДОЛЖАТ ДВИЖЕНИЕ, А ЛОГИКА СПАВНА ТАИЛОВ БУДЕТ ЖДАТЬ ПОКА ВСЯ ЦЕПОЧКА ИЗ ВСЕХ TASK ЗАКОНЧИТЬСЯ.

И ДА, ОТЧЕТ КООРДИНАТНОЙ СЕТКИ ИДЕТ НЕ ОТ ОСИ X, А ОТ ОСИ Z, ИЗ-ЗА ЧЕГО ПРИХОДИТЬСЯ ИСКАТЬ ТОЧКУ ПЕРЕСЕЧЕНИЯ ПЕРЕДОВАЯ X, Z, Y (А РЕЗУЛЬТАТ ЗАПИСЫВАТЬ В ОБРАТНОМ ПОРЯДКЕ) X=X, Y=Z, Z = Y

--

Движение реализовано через RigidBody, скрипт, который задает вектор направление движения талов (виден в инспекторе) и скрипта который по вектору направление, через заданное кол-во секунд добавляет скорость RigidBody

---

Так же, тут используется скрипт, который хранит в себе последнее найденное смещение (конечно не очень решение, но как пример отлично работать будет)

**Дополнительно**

(функционал, который можно убрать, он не влияет не на что)   
для каждого таила по 2 точкам(можно и больше) вычисляться границы таила.

(отдельно) Затем на основе границ каждого таила, вычисляется границы всей карты, состоящей из таилов

(!ВАЖНО!) И т.к. пока нету логики спавна стартовых таилов, первый таил в буфер(список) для расчета границ карты, нужно вносить ручками самому.

**V3**

V3 – в этой реализации нету точки спавна, т.к. она не нужна. Так же тут НЕ нужна идеальная точка пересечения. Тут у каждого таила есть скрипт, который содержит в себе точку, к которой будет присоединен следующий таил (поэтому отдельный префаб на этот таил Tile V3). Этот скрипт получаю через DKO таила. (при срабатывании триггера на спавн след. таила). После получения скрипта, просто беру эту точку, на неё установлю следующий таил.

--  
Движение реализовано через RigidBody, скрипт, который задает вектор направление движения талов (виден в инспекторе) и скрипта который по вектору направление, через заданное кол-во секунд добавляет скорость RigidBody

---

Так же, тут используется скрипт, который хранит в себе координаты точки установки следующего таила (полученные с последнего таила) (конечно не очень решение, но как пример отлично работать будет)

**!!ВАЖНО**. В ЭТОЙ РЕАЛИЗАЦИ НУЖНО БЫТЬ ОЧЕНЬ ОСТОРЖНЫМ СО ВСЕМИ TASK, Т.К. ЛЮБАЯ ЗАДЕРЖКАИ ИЗ-ЗА TASK, ОЗНАЧАЕТ, ЧТО НАЙДЕННОЕ СМЕЩЕНИЕ ДЛЯ СЛЕД, ТАИЛА СТАНЕТ НЕ АКТУЛЬНЫМ, Т.К. ТАИЛЫ ПРОДОЛЖАТ ДВИЖЕНИЕ, А ЛОГИКА СПАВНА ТАИЛОВ БУДЕТ ЖДАТЬ ПОКА ВСЯ ЦЕПОЧКА ИЗ ВСЕХ TASK ЗАКОНЧИТЬСЯ.

**Дополнительно**

(функционал, который можно убрать, он не влияет не на что)   
для каждого таила по 2 точкам(можно и больше) вычисляться границы таила.

(отдельно) Затем на основе границ каждого таила, вычисляется границы всей карты, состоящей из таилов

(! ВАЖНО!) И т.к. пока нету логики спавна стартовых таилов, первый таил в буфер(список) для расчета границ карты, нужно вносить ручками самому.

**V4**

V4 – это V2, но тут вместо специального скрипта триггера на спавн таилов с ДОП логикой, реализовал скрипта триггера на спавн таилов со список Task, которые должны быть выполнены перед оповещением об том что пора спавнить новый таил.

А логика нахождения точки пересечения двух векторов, была вынесена в отдельную Task. НО т.к. данных об колайдере я в Task не имею (только DKO таила), то пришлось сделать отдельную Task на таиле, которая возвращает координаты коллайдера(по этому отдельный префаб Tile V4)

**V5**

V3 – это V3, но тут вместо специального скрипта триггера на спавн таилов с ДОП логикой, реализовал скрипта триггера на спавн таилов со список Task, которые должны быть выполнены перед оповещением об том что пора спавнить новый таил.

А логика получения координат точки установки, следующего таила была вынесена в отдельную Task

**V6**

V6 – это V4, но тут используется пул для таилов. А раз есть пул, значит таилы будут переиспользоваться (отключаться и включаться), а значит у них должна быть такая логика, которая позволяла бы переиспользовать таил.

Поэтому в логике таилов добавляется 2 списка действий

1. Это список действий для активации таилов
2. Это список действий для дезактивации таилов

Теперь что касается самой работы с пулом. У логики пула есть 2 особенных интерфейса (не обязательных).

1. Нужен что бы сообщит элементу пула(в данном случае таилу), о том что он помещается в пул(а значит таил отработал и его нужно дезактивировать)  
   И на оборот, нужно сообщит элементу пула, о том, что этот элемент удаляется из пула(а значит званого используется, и этот таил нужно включить)
2. Это event который сообщает пулу о том, что этот объект возвращается в пул  
   Это нужна в случае, если элемент пула будет полностью самостоятелен и сам определять, что ему пора возвращаться в пул  
     
   Пр. пуля со своей уник. логикой, через 3 столкновения будет возвращаться в пул(в данном случае, пуля полностью сама будет вести счет об кол-ве столкновении и сообщать пулу через event, о том что ей нужно вернуться в пул)

Что бы эти event работали, скрипт, который будет храниться в пуле НЕ ДОЛЖЕН БЫТЬ ОТКЛЮЧЕН, даже если сам таил не используется (валяется в пуле).

Поэтому было принято решение, что класс, который будет храниться в пуле, это НЕ ТОТ ЖЕ КЛАСС, ЧТО И КЛАСС ПРЕФАБА.

Это дополнительно дает модульность и видимость использует ли таил пул или нет.

Скрипт пула с таила будем получать через DKO таила.

Спавн таилов происходит через пул

Смерть таилов (их отключение и возвращение в пул) происходит в 2 вариантах (оба есть на сцене для примера)

В1) Это передаем в пул напрямую объект, который будет возвращен в пул

В2) Это вызываем возращение в пул объекта через его event  
 (при такой реализации объект конечно не полностью самостоятелен получается, но как наглядный пример сойдет)

**V7**

V7 – это V6, Но теперь тут у таилов есть типы, и пул тоже по типу используется.

Раз есть типы таилов, значит таилы должны чем-то различаться. В данной реализации для примера есть 2 основных типа

1. Это таилы которые используют систему пула (имеют логику для пера использования)
2. Это обычные одноразовые таилы

У этих типов различаться способы спавна. Для тех, которые используют пул, спавн происходит через пул. А те, которые обычные, просто спавняться.

В логики смерти (отключение таилов) тоже есть различия, для тех которые используют пул есть 2 варианта (как в V6) передовая объект сразу в пул или через event у таила.

А у талов обычных, просто полностью отключаю таил навсегда. (и этот отключенный таил остаётся на сцене пока не закончиться игра или пока не произойдёт переход на другую сцену)

Так же у таилов которые по типу есть отличие в классе, в виде того, что он должен получать тип тала. То есть таил хранит в себе свой тип. Он устанавливается при спавне таила. Является отдельным скриптом. Обернут через DKO отдельно для Get и Set, чтобы инкапсулировать доступ.

**Так же в случае стартовых таилов. У них задаю ключ заранее через инспектора**

Так же тут есть логика выбора следующего типа таила (в аргументы передаться последний известный тип таила). В данной реализации используется логика рандомного выбора следующего типа.

НО, на можно спокойно написать логику выбора след. типа, на основе предыдущего (что бы так сделать целую цепочку из которой будет сделана карта с таилами)

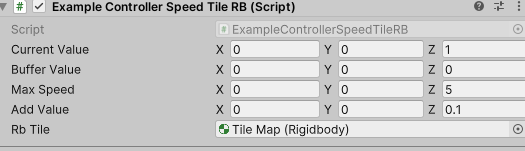
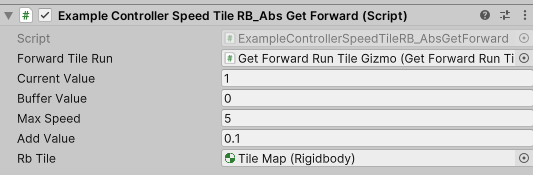
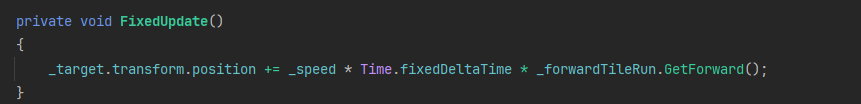
Пр. Сначала препятствие, потом таил с бонусом, потом простой таил, затем препятствие и наконец таил с наградой (пр. монетами) и затем званого цепочку повторять.

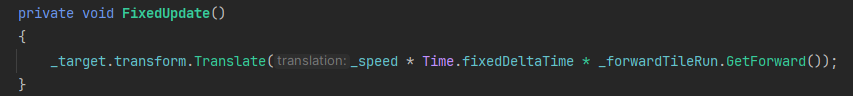
**А т.к в одном типе целый список префабов, то и препятствия могут быть разные и бонусы и т.д.**

Так же есть система выбора префаб из списка префабом. В данной реализации тоже реализовано через рандом.

**Про виды движения таилов**

(Логика движения таилов полностью отделена от самой логики таилов)

1. Используя физику(RigidBody)
   1. Это скрипт. Где вручную задаться вектор направление движения таилов, начальное значение скорости передвижения таилов, максимальное значение таилов и значение которое будет прибавляться для движения таилов. И все это отдельно для каждой оси(X, Y, Z)  
      
   2. Это скрипт. Где, начальное значение скорости передвижения таилов, максимальное значение таилов и значение которое будет прибавляться для движения таилов. А вектор направление таилов определяет отдельный скрипт   
      
2. Не используя физику. Через FixUpdate()  
    Используя вектор направления движения таилов находим следующую точку где бы находился таил через указанный промежуток времени умноженный на скорость движения таила(и все это прибавляется тупо к текущей координате таила)  
   можно использует   
   а можно используя.



Это полностью одинаковые записи (метод Translate тупо добавляет указанное значение к текущей координате (максимально бесполезный метод))