**1. Создание проекта и материалы**

-создаем проект

-ресурсы проекта (---), перенос проектов

-создание простого цилиндра, за счет Префаба

-создаем пустой объект Tower (Спавнер, Вращение, Платформы, Шарик)

**2. Начало спаунера**

-создаем скрипт

|  |
| --- |
| TowerBilder.cs |
| public class TowerBilder : MonoBehaviour  {  //количество уровней (платформ сверху-вниз)  [SerializeField] private int \_levelCount;  //то на чем будет держатся платформы  [SerializeField] private GameObject \_beam;  //то, что мы будем спавнить  //т.к. нам нужно, чтобы Цилиндр увеличивался только вниз(считаем)  /// <summary>  /// < Начало постройки башни  /// </summary>  private void Start()  {  Build();  }  /// <summary>  /// Постройка Башни  /// </summary>  private void Build() {  //Спавним объект как родительский класс Tower, при этом заметим,  //что нам необходимо ставить каждый раз его ниже  GameObject beam = Instantiate( \_beam, transform);  beam.transform.localScale = new Vector3(1, \_levelCount / 2f ,1);  }  } |

**3. Создаем платформы**

-создаем несколько моделей платформ

1) Финишная

2) Стартовая

3) Простая

**-накидываем на неё компонент MechCollider и ставим галочку на Convex**

-создаем Типизацию объектов на сцене, для этого создаем отдельные скрипты для каждого объекта, а в главном коде прописываем типизацию (не забываем задать скрипты для Префабов)

|  |
| --- |
| TowerBilder.cs |
| public class TowerBilder : MonoBehaviour  {  [SerializeField] private int \_levelCount;  [SerializeField] private GameObject \_beam;  [SerializeField] private SpawnPlatform \_spawnPlatform;  [SerializeField] private Platform[] \_Platform;  [SerializeField] private FinichPlatform \_finichPlatform;  } |
|  |

**4. Спаун платформ**

|  |
| --- |
| TowerBilder.cs |
| private void Build() {  GameObject beam = Instantiate(\_beam, transform);  beam.transform.localScale = new Vector3(1, \_levelCount / 2f ,1);  Vector3 spawnPosition = beam.transform.position;  ///< получение самой верхней точки  spawnPosition.y += beam.transform.localScale.y;  for (int i = 0; i < \_levelCount; i++)  {  ///в зависимости от видов платформ выбирается одна  SpawnPlatform(  \_Platform[Random.Range(0, \_Platform.Length)],  ref spawnPosition,  beam.transform);  }  }  private void SpawnPlatform(Platform platform, ref Vector3 SpawnPosition, Transform parent) {  Instantiate(platform, SpawnPosition, Quaternion.Euler(0, Random.Range(0, 360), 0), parent);  SpawnPosition.y -= 1;  } |

-немного редактируем модели (т.к. мои модели «собственные)

**5. Стартовая и финишные платформы**

-на стартовой есть Создается Шарик

-на финишной заканчивается уровень

|  |
| --- |
| TowerBilder.cs |
| //Scale в учет нашего финиша и старта  private float startAndFinifAdditionalSceale = 0.5f;    public float BeamScaleY => \_levelCount / 2f + startAndFinifAdditionalSceale;  private void Build() {  beam.transform.localScale = new Vector3(1, BeamScaleY, 1);  } |
| Реализуем создание Стартовой Платформы -используем Наследование |
| public class SpawnPlatform : Platform{  } |
| private void Build() {  GameObject beam = Instantiate(\_beam, transform);  beam.transform.localScale = new Vector3(1, BeamScaleY, 1);  Vector3 spawnPosition = beam.transform.position;  spawnPosition.y += beam.transform.localScale.y;  SpawnPlatform(\_spawnPlatform, ref spawnPosition, beam.transform);  for (int i = 0; i < \_levelCount; i++){  SpawnPlatform(  \_Platform[Random.Range(0, \_Platform.Length)],  ref spawnPosition,  beam.transform);  }  SpawnPlatform(\_finichPlatform, ref spawnPosition, beam.transform);  } |

-в начале у нас обрубок, исправляем

|  |
| --- |
| TowerBilder.cs |
| [SerializeField] private float \_additionalScale;//задаем самостоятельно  private void Build() {  spawnPosition.y += beam.transform.localScale.y - \_additionalScale;  } |

**6. Вращение Башни**

-добавляем нашему элементу Tower RigidBody2D(для плавного вращения)

|  |
| --- |
|  |
| При чем ставим 1, для плавности |

-далее переходим в BuildSettings и Выставляем Android

|  |
| --- |
| TowerRotater.cs |
| [RequireComponent(typeof(Rigidbody))]  public class TowerRotater : MonoBehaviour  {  [SerializeField] private float \_rotateSpeed = 5;  private Rigidbody rigidbody;  private void Start()  {  rigidbody = GetComponent<Rigidbody>();  }  private void Update()  {  //есть есть палец  if (Input.touchCount > 0){  //получаем палец  Touch touch = Input.GetTouch(0);  //если палец двигается  if (touch.phase == TouchPhase.Moved){  //считаем как далеко двинулся и рассчитываем  float torgue = touch.deltaPosition.x \* Time.deltaTime \* \_rotateSpeed;  rigidbody.AddTorque(Vector3.up \* torgue);  }  }  }  } |

**7. Создание шарика**

|  |
| --- |
|  |
| Замораживаем все лишние вращения |

-создаем простой элемент Сфера, далее подбираем размер фигуры и замораживаем не нужные для нас позиции и вращения

|  |
| --- |
|  |
| -может только прыгать |
| Внимание! Взаимодействие коллайдеров родителей и дочерних объектов |

|  |
| --- |
| BallJumper.cs |
| [RequireComponent(typeof(Rigidbody))]  public class BallJumper : MonoBehaviour{  [SerializeField] private float \_jumpForce = 5;  private Rigidbody rigidbody;  private void Start() {  rigidbody = GetComponent<Rigidbody>();  }  private void OnCollisionEnter(Collision collision) {  if (collision.gameObject.TryGetComponent(out PlatformSegment platformSegment)){  rigidbody.AddForce(Vector3.up \* \_jumpForce, ForceMode.Impulse);  }  }  } |

-добавляем скрипт, для отслеживания касания по Сегменту платформы

|  |
| --- |
| PlatformSegment.cs |
|  |
| Для каждого куска платформы |
| Повторяем этот же момент и для SpawnPlatform |
|  |

-удаляем коллайдер для нашего цилиндра, чтобы он не ускорялся

**8. Спаун Шарика**

|  |
| --- |
| SpawnPlatform.cs |
| [SerializeField] private Ball \_ball;  [SerializeField] private Transform \_spawnPoint;  private void Awake()  {  Instantiate(\_ball, \_spawnPoint.position, Quaternion.identity);  } |
| Используем Awake, чтобы до запуска уровня, наша камера уже была на нашем Шарике |

|  |
| --- |
| Ball.cs |
| public class Ball : MonoBehaviour  {  } |
| -необходим для типизации нашего Шарика |

-создаем в префабе SpawnPlatform пустой объект, и размещаем его положения, для спавна

|  |
| --- |
| BallJumper.cs |
| private void OnCollisionEnter(Collision collision)  {  if (collision.gameObject.TryGetComponent(out PlatformSegment platformSegment))  {  rigidbody.velocity = Vector3.zero;  rigidbody.AddForce(Vector3.up \* \_jumpForce, ForceMode.Impulse);  }  } |
| -немного меняем код, для более четкого прыжка шарика |

**9. Разрушение платформ**

-полностью доделываем все наши платформы (однако добавляем незаконченным частям триггер и удаляем у них MeshRender), при чем тригерную платформу необходимо немного уменьшить (чтобы лишний раз не засчитывала уровень)

|  |
| --- |
|  |
|  |

|  |
| --- |
| PlatformSegment.cs |
| [RequireComponent(typeof(Rigidbody))]  public class PlatformSegment : MonoBehaviour  {  public void Bounce(float Force, Vector3 centre, float raduis) {  if (TryGetComponent(out Rigidbody rigidbody)){  rigidbody.isKinematic = false;  rigidbody.AddExplosionForce(Force, centre, raduis);  }    }  } |
| -нам необходимо при прохождении уровне, пробежаться по всем частям платформы, убрать Кинемтатику и толкнуть их посильнее |

|  |
| --- |
| Platform.cs |
| public class Platform : MonoBehaviour  {  [SerializeField] private float BounceForce;  [SerializeField] private float BonceRadius;  public void Break()  {  PlatformSegment[] PlatformSegment = GetComponentsInChildren<PlatformSegment>();  foreach (var segment in PlatformSegment) {  segment.Bounce(BounceForce, transform.position, BonceRadius);  }  }  } |
| Необходим для определения расталкивания частиц(платформ)  -при чем выполняем для каждого кусочка дочерних класссов |

|  |
| --- |
| Ball.cs |
| public class Ball : MonoBehaviour  {  private void OnTriggerEnter(Collider other)  {  if (other.TryGetComponent(out PlatformSegment platform)){  other.GetComponentInParent<Platform>().Break();  }  }  } |
| Проверяет, столкнулись ли мы с Триггером, если да, то мы идем к родителю и вызываем его метод |

-не забываем настроить нашим плафтормам (PlatformerSegment) и по 50 силу и радиус

**10. Отслеживающая камера**

|  |
| --- |
| BallTracing.cs |
| public class BallTracing : MonoBehaviour  {  [SerializeField] private Vector3 directionOffset;  [SerializeField] private float length;  private Ball ball;  private Beam beam;  private Vector3 camerPosition;  private Vector3 minimemBallPosistion;    private void Start() {  ball = FindObjectOfType<Ball>();  beam = FindObjectOfType<Beam>();  TrackBall();  }  private void Update(){  if (ball.transform.position.y < minimemBallPosistion.y) {  TrackBall();  minimemBallPosistion = ball.transform.position;  }  }  private void TrackBall() {  camerPosition = ball.transform.position;  Vector3 direction = (beam.transform.position - ball.transform.position).normalized + directionOffset;  camerPosition -= direction \* length;  transform.position = camerPosition;  transform.LookAt(ball.transform);  }  } |
| Получение направление шарика и цилиндра, при чем нормализованное. Далее мы его поправляем. |
| Находим в сцене объекты. При чем создаем несколько переменных, для отображения камеры по положениям (ПозицияКамеры, Минимальная) |
| Для того, чтобы код работал правильно, а именно функция Start(), необходимо в скрипте TowerBilder.cs поменять Start на Awake |
| Однако при расчете у нас возникает ошибка, т.к. считает по центру из-за этого камера переворачивается, делаем хитрость |
| private void TrackBall()  {  Vector3 beamPosition = beam.transform.position;  beamPosition.y = ball.transform.position.y;  camerPosition = ball.transform.position;  Vector3 direction = (beamPosition - ball.transform.position).normalized + directionOffset;  camerPosition -= direction \* length;  transform.position = camerPosition;  transform.LookAt(ball.transform);  } |

|  |
| --- |
| Beam.cs |
|  |

**11. Настраиваем отображение и цвета**

-Для этого создаем отдельную папку Materials, создаем Материалы, для Ball, Beam, Platform

-Задаем им понравившиеся цвета из сайта colors.co

-И цепляем их к объектам

|  |
| --- |
|  |

-переходим в окно работы со светом

|  |
| --- |
|  |
|  |
|  |

Так же убираем эффект тени со света

|  |
| --- |
|  |

Делаем фон одноцветным

|  |
| --- |
|  |

Делаем наши Материалы Матовыми

|  |
| --- |
|  |