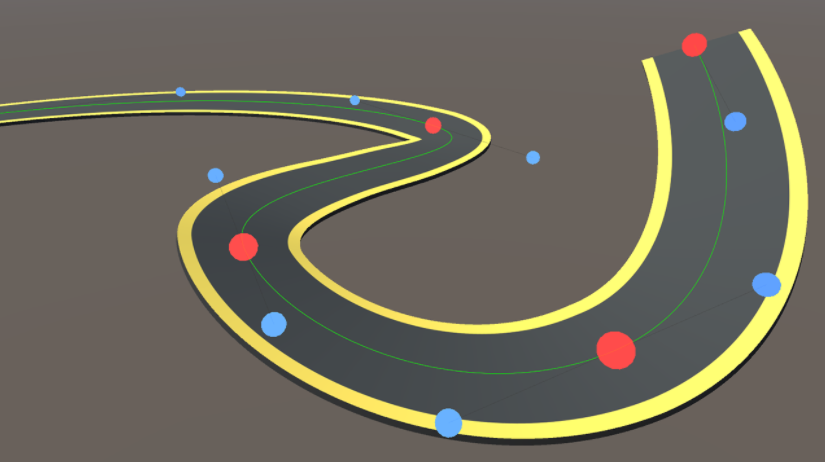
**1. Ассеты**

Для данного проекта нам понадобятся внедрить один Ассет (модели Людей) и плагин, который рисует кривые Безье.

Ассет моделей, скачанных мной [Toony Tiny People Demo]:



Ссылка на GitHub, для генерации дороги [https://github.com/SebLague/Path-Creator]:

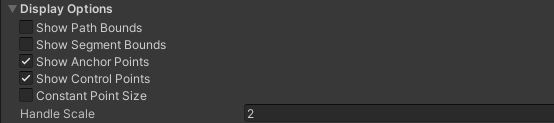


**2. Создание дороги**

Все необходимые методы и принципы работы с данной библиотекой указаны в технической документации.

Для создания дороги создаем Пустой Объект и подключаем компонент PathCreator(настройка дороги) и RoadMeshCreator(отображение дороги) и выставляем следующие свойства.

Для удобства настройка Сфер выставляем Handle Scale

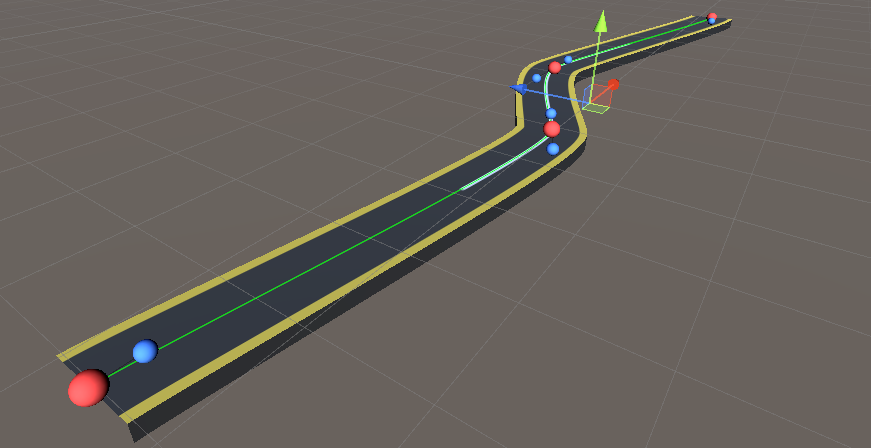


Для добавления дополнительной части дороги добавляем shift + Left Click, и настраивая синие и красные сферы реализуем дорогу.

Для отображения корректной дороги необходимо в параметре BezeirPathOptions Space как Top-down(xz).



В результате манипуляций получится следующая дорога:



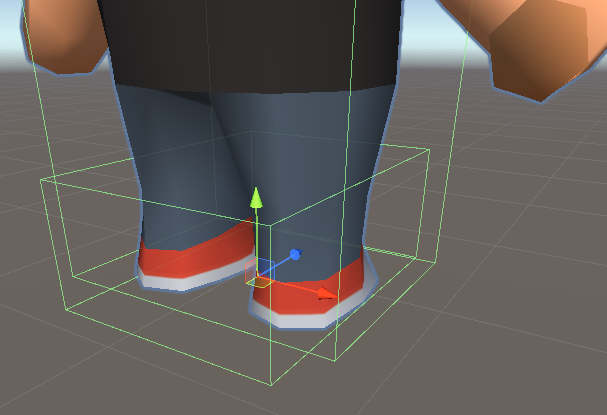
**3. Следование по пути**

Реализуем через код и взаимодействия с библиотекой.   
Для начала добавляем Персонажа (из Компонентов Transfom, Animator). Пишем код для нашего персонажа и не забываем указать параметры в Unity.

|  |
| --- |
| PathFollower.cs |
| using PathCreation;  [RequireComponent(typeof(Rigidbody))]  public class PathFollower : MonoBehaviour{  [SerializeField] private float speed;  [SerializeField] private PathCreator pathCreator;  private Rigidbody rigidbodyMan;  private float distanceTravelled;  private void Start() {  rigidbodyMan = GetComponent<Rigidbody>();  rigidbodyMan.MovePosition(pathCreator.path.GetPointAtDistance(distanceTravelled));  }  private void Update() {  distanceTravelled += Time.deltaTime \* speed;  Vector3 nextPoint = pathCreator.path.GetPointAtDistance(distanceTravelled, EndOfPathInstruction.Loop);  nextPoint.y = transform.position.y;  transform.LookAt(nextPoint);  rigidbodyMan.MovePosition(nextPoint);  }  } |

Для правильной работы данного кода необходимо для нашего персонажа и для нашей дороги добавить BoxCollider и Mesh Collider соответственно.

Для последующей механики игры, создаем пустой объект Tower и делаем нашего персонажа дочерним к объекту. Так же накидываем отдельно Коллайдер для нашего персонажа и для нашего Tower.



**4. Прыжки**

Для прыжков создаем новый скрипт и кидаем его на объект Tower. При этом реализуем логику, когда наш персонаж не должен прыгать несколько раз, через CollisionEnter. Настраиваем силу прыжка.

|  |
| --- |
| Jumper.cs |
| public class Jumper : MonoBehaviour{  [SerializeField] private float jumpForce;  private Rigidbody rirgidbody;  private bool isgrounded;  private void Start() {  rirgidbody = GetComponent<Rigidbody>();  }  private void Update() {  if (Input.GetMouseButtonDown(0) && isgrounded == true) {  isgrounded = false;  rirgidbody.AddForce(Vector3.up \* jumpForce);  }  }  private void OnCollisionEnter(Collision collision) {  if (collision.gameObject.TryGetComponent(out Road road)){  isgrounded = true;  }  }  } |

**Необходимо будет реализовать увеличение/уменьшении силы прыжка, при добавлении новых персонажей.**

**5. Подготовка людей для башен**

Для генерации уровня и игры мы создаем отдельный объект Tower, а основной игровой объект переименуем в PlayerTower.

Для постройки нам необходимы точки, где будет строятся наши башни. Для этого выбираем несколько Моделей и убираем всё лишнее, кроме компонента Animator и настраиваем им дополнительный компонент BoxCollider. Добавляем Людей новый скрипт Human.cs, для типизации объектов.

Для генерации точки, мы в Префабе создаем точку на плечах FixedPoint.



В скритпе Human.cs реализуем **только получение точки**:

|  |
| --- |
| Human.cs |
| public class Human : MonoBehaviour  {  [SerializeField] private Transform fixationpoint;  public Transform FixationPoint => fixationpoint; //(можно только читать)  } |

Добавляем в срипт нашу точку.

После добавляем данную точку, для каждого Префаба.

**6. Башни**

Для реализации Башни из Людей напишем скрипт:

|  |
| --- |
| Tower.cs |
| public class Tower : MonoBehaviour  {  [SerializeField] private Vector2Int humanInTowerRange;  [SerializeField] private Human[] humansTemplates;  private List<Human> humanInTower;  private void Start(){  humanInTower = new List<Human>();  int humaninTowerCount = Random.Range(humanInTowerRange.x, humanInTowerRange.y);  SpawnHumans(humaninTowerCount);  }  private void SpawnHumans(int humanCount) {  Vector3 spawnPoint = transform.position;  for (int i = 0; i < humanCount; i++) {  Human spawnedHuman = humansTemplates[Random.Range(0, humansTemplates.Length)];  humanInTower.Add(Instantiate(spawnedHuman, spawnPoint, Quaternion.identity, transform));  humanInTower[i].transform.localPosition = new Vector3(0, humanInTower[i].transform.localPosition.y, 0);  spawnPoint = humanInTower[i].FixationPoint.position;  }  }  } |
| Особенности Кода:  -рандомный диапазон из Vector2Int  -массив Префабов Human[] |

Добавляем скрипт на наш объект Tower и вставляем параметры.

**7. Генерация уровня**

Для реализации Генерации Уровня пишем следующий скрипт и добавляем его на объект Path:

|  |
| --- |
| LevelCreator.cs |
| using PathCreation; public class LevelCreator : MonoBehaviour{  [SerializeField] private PathCreator pathCreator;  [SerializeField] private Tower TowerTemplate;  [SerializeField] private int humanTowerCount;  private void Start(){  GenerateLevel();  }  private void GenerateLevel() {  float roadLength = pathCreator.path.length;  float distanceBetweenTower = roadLength / humanTowerCount;  float distanceTravelled = 0;  Vector3 spawnPoint;  for (int i = 0; i < humanTowerCount; i++) {  distanceTravelled += distanceBetweenTower;  spawnPoint = pathCreator.path.GetPointAtDistance(distanceTravelled, EndOfPathInstruction.Stop);  Instantiate(TowerTemplate, spawnPoint, Quaternion.identity);  }  }  } |
| Особенности Кода: |

**8. Подбираем людей**

Далее в игре т.к. взаимодействие реализовано через Collision нам необходимо чтобы физика не просчитывала определенные взаимодействия между различными типами объектов или «Слоями/Lauers».  
Для этого реализуем следующее:  
1) Выделяем для них слой Human (Выбираем Yes,Children)  
2) Переходим в Project Settings/Physics  
3) В матрице взаимодействия физики убираем галочку с Human/Human

Для того, чтобы убрать другие лишние взаимодействия можно выставить в BoxCollider isTrigger.

Создаем скрипт PlayerTower.cs и навешаем на наш объект. (Наша основная задача при прыжке определить кого мы коснулись и высчитать расстояние для определения какое число людей добавляется на нашу башню):   
-для проверки расстояние у игрока создаем пустой объект Checker на ногах

|  |
| --- |
| PlayerTower.cs |
| public class PlayerTower : MonoBehaviour{  [SerializeField] private Human StartHuman;  [SerializeField] private Transform distanceChecker;  [SerializeField] private float fixationMaxDistance;  [SerializeField] private BoxCollider checkCollider;  private List<Human> humans;  /// Создаение Человека  private void Start(){  humans = new List<Human>();  Vector3 spawnPoint = transform.position;  humans.Add(Instantiate(StartHuman, spawnPoint, Quaternion.identity, transform));  }  /// Обработка столкновений  /// При столкновении и проверки мы берем всех нижних Людей  private void OnTriggerEnter(Collider other) {  if (other.gameObject.TryGetComponent(out Human human)){  Tower collisionTower = human.GetComponentInParent<Tower>();  if (other != null) {  List<Human> collectedhumans = collisionTower.CollectHuman(distanceChecker, fixationMaxDistance);  if (collectedhumans != null) {  for (int i = collectedhumans.Count - 1; i >= 0; i--) {  Human insertHuman = collectedhumans[i];  InsetrHuman(insertHuman);  //DisplaysChecker(insertHuman); (9-часть)  } } } } }  private void InsetrHuman(Human collectedhumans) {  humans.Insert(0, collectedhumans);  SetHumanPoisition(collectedhumans);  }  private void SetHumanPoisition(Human human) {  human.transform.parent = transform;  human.transform.localPosition = new Vector3(0, human.transform.localPosition.y, 0);  human.transform.localRotation = Quaternion.identity;  } |

Далее в скрипт Tower.cs добавляем два дополнительных метода, для обработки и передачи количество элементов в башне (Первый возвращаем лист Элементов, Второй проверяет расстояние по ОсиY):

|  |
| --- |
| Tower.cs |
| /// Получение элементов  public List<Human> CollectHuman(Transform distanceChecker, float fixationMaxDistance) {  for (int i = 0; i < humanInTower.Count; i++){  float distanceBetweenPoints = CheckDistanceY(distanceChecker, humanInTower[i].FixationPoint.transform);  if (distanceBetweenPoints < fixationMaxDistance) {  List<Human> collectedHumans = humanInTower.GetRange(0, i + 1);  humanInTower.RemoveRange(0, i + 1);  return collectedHumans;  }  }  return null;  }  /// Получение дистанции  private float CheckDistanceY(Transform distanceChecker,Transform humanFixationPoint) {  Vector3 distanceCheckerY = new Vector3(0, distanceChecker.position.y, 0);  Vector3 humanFixationPointY = new Vector3(0, humanFixationPoint.position.y, 0);  return Vector3.Distance(distanceCheckerY, humanFixationPointY);  } |

После добавления скрипта у нас небольшая ошибка, наш Чекер остается в ногах нижнего элемента.

**9. Смещаем чекеры**

Реализуем в скоипте PlayerTower.cs смещение чекера:

|  |
| --- |
| PlayerTower.cs |
| private void DisplaysChecker(Human human) {  float displaceScale = 1.5f;  Vector3 distanceChecerNewPosition = distanceChecker.position;  distanceChecerNewPosition.y -= human.transform.localScale.y \* displaceScale;  distanceChecker.position = distanceChecerNewPosition;  checkCollider.center = distanceChecker.localPosition;  } |

Добавляем слой для нашего PlayerTower (Внимательно! Только на объект No, this object Only) и в матрице взаимодействия убираем галочку взаимодействия с нашей дорогой Road.

**10. Ломаем башни**

В скрипте Tower.cs добавляес скрипт

|  |
| --- |
| Tower.cs |
| public void Break()  {  //rigidbody=добавляем iskinematic=false parent=null  Destroy(gameObject);  } |
| **Домашнее задание на реализацию** |

Вызов метода в PlayerTower.cs

|  |
| --- |
| PlayerTower.cs |
| private void OnTriggerEnter(Collider other) {  if (other.gameObject.TryGetComponent(out Human human)){  Tower collisionTower = human.GetComponentInParent<Tower>();  if (other != null){  List<Human> collectedhumans = collisionTower.CollectHuman(distanceChecker, fixationMaxDistance);  if (collectedhumans != null){  for (int i = collectedhumans.Count - 1; i >= 0; i--){  Human insertHuman = collectedhumans[i];  InsetrHuman(insertHuman);  DisplaysChecker(insertHuman);  }  }  collisionTower.Break();  }  }  } |

**11. Рекомендации по ДЗ**

ДЗ на собственную реализацию:

1. Усилитель прыжков
2. Анимация разрушения башни
3. Препятствия

**12. Анимация бега для первого человека**

У каждого Человека есть Animator мы создаем для ней Контроллер. Далее клип выставляем в наше поле.

Создаем переменные для в Animator для изменения состояние анимации, перехода между состояниями.

В скрипте Human.cs реализуем управление состоянием анимации:

|  |
| --- |
| Human.cs |
| [RequireComponent(typeof(Animator))]  public class Human : MonoBehaviour{  [SerializeField] private Transform fixationpoint;  public Transform FixationPoint => fixationpoint; //(можно только читать)  private Animator animator;  private void Awake() {  animator = GetComponent<Animator>();  }  public void Run() {  animator.SetBool("isRun", true);  }  public void StopRun() {  animator.SetBool("isRun", false);  }  } |

В переходе между клипами есть Make Transition, где выставляем необходимые условия перехода.

**13. Отслеживание башни**

Настройка камеры через скрипт PlayerTracker.cs, но добавим камеру как дочерним объект к PlayerTower.

|  |
| --- |
| PlayerTracker.cs |
| public class PlayerTracker : MonoBehaviour{  [SerializeField] private PlayerTower playerTower;  [SerializeField] private Vector3 offsetPoisiton;  [SerializeField] private Vector3 offsetRotation;  [SerializeField] private float moveSpeed;  private Vector3 targetOffsetPosition;  private void Start() {  targetOffsetPosition = playerTower.transform.position + offsetPoisiton;  }  private void Update() {  UpdatePoisition();  targetOffsetPosition = Vector3.MoveTowards(offsetPoisiton, targetOffsetPosition, moveSpeed \* Time.deltaTime);  }  private void UpdatePoisition() {  transform.position = playerTower.transform.position;  transform.localPosition += offsetPoisiton;  Vector3 lookAtPoint = playerTower.transform.position + offsetRotation;  transform.LookAt(lookAtPoint);  }  } |

**14. Отдаляем камеру при добавлении людей**

Реализуем через событие:

|  |
| --- |
| PlayerTower.cs |
| public class PlayerTower : MonoBehaviour{  ////  public event UnityAction<int> HumanAdded;  private void Start(){  ////  HumanAdded?.Invoke(humans.Count);  }  private void OnTriggerEnter(Collider other) {  if (other.gameObject.TryGetComponent(out Human human)){ //  if (other != null){//  if (collectedhumans != null){//  for (int i = collectedhumans.Count - 1; i >= 0; i--) {//  }  HumanAdded?.Invoke(humans.Count);  humans[0].Run();  }  collisionTower.Break();  }  }  } |

Добавляем дополнительный метод для PlayerTracker.cs:

|  |
| --- |
| PlayerTracker.cs |
| private void OnEnable() {  playerTower.HumanAdded += OnHumanAdded;  }  private void OnDisable() {  playerTower.HumanAdded -= OnHumanAdded;  } private void OnHumanAdded(int count) {  targetOffsetPosition += offsetPoisiton + (Vector3.up + Vector3.back) \* count/2;  UpdatePoisition();  } |

**15. Плавное изменение позиции камеры**

Реализовано в 13-14 части занятия.

**16.Украшаем**

Без украшений.