Ropebound Ninja Rampage

1.0

Generated by Doxygen 1.11.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 advancedClimbing Class Reference

This class handles advanced climbing mechanics, including line movement and ziplining.

Inherits MonoBehaviour.

Public Attributes

- Camera cam

  Reference to the Camera component.
- float afterHandleJumpForce

  Force applied after jumping off a handle.
- float sphereCastR

  Radius of the sphere cast used for detection.
- float hitingLenght

  Length of the hit detection.
- float handleAfterJumpDelay

  Delay after jumping off a handle.
- LayerMask Handler

  LayerMask for detecting handlers.
- bool canHandle

  Indicates if the player can handle.
- bool stopHolding

  Indicates if the player should stop holding.

Private Member Functions

- void Start ()

  Initializes the necessary components and variables.
- void Update ()

  Updates the climbing logic each frame.
- void OnCollisionEnter (Collision other)

  Called when the collider enters a collision.
- void OnCollisionExit (Collision other)

  Called when the collider exits a collision.

Private Attributes

- Rigidbody rigidbody

  Reference to the Rigidbody component.
- RaycastHit hit

  RaycastHit for detecting collisions.
- LineMovement lineMovement

  Reference to the LineMovement component.
- Ziplining ziplining

  Reference to the Ziplining component.

4.1.1 Detailed Description

This class handles advanced climbing mechanics, including line movement and ziplining.

Definition at line 6 of file advancedClimbing.cs.

4.1.2 Member Function Documentation

4.1.2.1 OnCollisionEnter()

```
void advancedClimbing.OnCollisionEnter (
            Collision other)  [private]
```

Called when the collider enters a collision.

Parameters

other	The Collision data associated with this collision.

Definition at line 85 of file advancedClimbing.cs.

4.1.2.2 OnCollisionExit()

```
void advancedClimbing.OnCollisionExit (
            Collision other)  [private]
```

Called when the collider exits a collision.

Parameters

other	The Collision data associated with this collision.

Definition at line 115 of file advancedClimbing.cs.

4.1.2.3 Start()

```
void advancedClimbing.Start ()  [private]
```

Initializes the necessary components and variables.

Definition at line 41 of file advancedClimbing.cs.

4.1.2.4 Update()

```
void advancedClimbing.Update ()  [private]
```

Updates the climbing logic each frame.

Definition at line 54 of file advancedClimbing.cs.

4.1.3 Member Data Documentation

4.1.3.1 afterHandleJumpForce

```
float advancedClimbing.afterHandleJumpForce
```

Force applied after jumping off a handle.

Definition at line 16 of file advancedClimbing.cs.

4.1.3.2 cam

```
Camera advancedClimbing.cam
```

Reference to the Camera component.

Definition at line 12 of file advancedClimbing.cs.

4.1.3.3 canHandle

```
bool advancedClimbing.canHandle
```

Indicates if the player can handle.

Definition at line 34 of file advancedClimbing.cs.

4.1.3.4 handleAfterJumpDelay

```
float advancedClimbing.handleAfterJumpDelay
```

Delay after jumping off a handle.

Definition at line 22 of file advancedClimbing.cs.

4.1.3.5 Handler

```
LayerMask advancedClimbing.Handler
```

LayerMask for detecting handlers.

Definition at line 24 of file advancedClimbing.cs.

4.1.3.6 hit

```
RaycastHit advancedClimbing.hit  [private]
```

RaycastHit for detecting collisions.

Definition at line 26 of file advancedClimbing.cs.

4.1.3.7 hitingLenght

```
float advancedClimbing.hitingLenght
```

Length of the hit detection.

Definition at line 20 of file advancedClimbing.cs.

4.1.3.8 lineMovement

```
LineMovement advancedClimbing.lineMovement  [private]
```

Reference to the LineMovement component.

Definition at line 29 of file advancedClimbing.cs.

4.1.3.9 rigidbody

```
Rigidbody advancedClimbing.rigidbody  [private]
```

Reference to the Rigidbody component.

Definition at line 10 of file advancedClimbing.cs.

4.1.3.10 sphereCastR

```
float advancedClimbing.sphereCastR
```

Radius of the sphere cast used for detection.

Definition at line 18 of file advancedClimbing.cs.

4.1.3.11 stopHolding

```
bool advancedClimbing.stopHolding
```

Indicates if the player should stop holding.

Definition at line 36 of file advancedClimbing.cs.

4.1.3.12 ziplining

```
Ziplining advancedClimbing.ziplining  [private]
```

Reference to the Ziplining component.

Definition at line 31 of file advancedClimbing.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/advancedClimbing.cs

4.2 AvalibleIfLevel Class Reference

This class controls the activation of a GameObject based on the player's current level.

Inherits MonoBehaviour.

Public Attributes

- int levelToActive
  The level required to activate the GameObject.

Private Member Functions

- void Start ()
  Initializes the GameObject and sets its active state based on the player's current level.

4.2.1 Detailed Description

This class controls the activation of a GameObject based on the player's current level.

Definition at line 6 of file AvalibleIfLevel.cs.

4.2.2 Member Function Documentation

4.2.2.1 Start()

```
void AvalibleIfLevel.Start ()  [private]
```

Initializes the GameObject and sets its active state based on the player's current level.

Definition at line 14 of file AvalibleIfLevel.cs.

4.2.3 Member Data Documentation

4.2.3.1 levelToActive

```
int AvalibleIfLevel.levelToActive
```

The level required to activate the GameObject.

Definition at line 9 of file AvalibleIfLevel.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/AvalibleIfLevel.cs

4.3 Camera3P Class Reference

This class controls the third-person camera behavior, including aiming and default camera modes.

Inherits MonoBehaviour.

Public Attributes

- Transform orientation
  Reference to the orientation transform.
- Transform player
  Reference to the player transform.
- Transform playerObject
  Reference to the player object transform.
- Camera cam
  Reference to the Camera component.
- float rotationSpeed
  Speed of camera rotation.
- Transform lookDir
  Direction the camera is looking at.
- bool aim
  Indicates if aiming mode is active.
- Image aimImage
  UI Image to display when aiming.
- bool disableRotation
  Indicates if rotation is disabled.
- CinemachineFreeLook defaultCinemachine
  Default Cinemachine camera for third-person view.
- CinemachineFreeLook aimCinemachine
  Cinemachine camera for aiming view.

Private Member Functions

- void Start ()

  Initializes the camera settings and variables.
- void Update ()

  Updates the camera behavior each frame.

Private Attributes

- float horizontalI

  Horizontal input value.
- float verticalI

  Vertical input value.

4.3.1 Detailed Description

This class controls the third-person camera behavior, including aiming and default camera modes.

Definition at line 8 of file Camera3P.cs.

4.3.2 Member Function Documentation

4.3.2.1 Start()

```
void Camera3P.Start ()  [private]
```

Initializes the camera settings and variables.

Definition at line 48 of file Camera3P.cs.

4.3.2.2 Update()

```
void Camera3P.Update ()  [private]
```

Updates the camera behavior each frame.

Definition at line 59 of file Camera3P.cs.

4.3.3 Member Data Documentation

4.3.3.1 aim

```
bool Camera3P.aim
```

Indicates if aiming mode is active.

Definition at line 33 of file Camera3P.cs.

4.3.3.2 aimCinemachine

```
CinemachineFreeLook Camera3P.aimCinemachine
```

Cinemachine camera for aiming view.

Definition at line 43 of file Camera3P.cs.

4.3.3.3 aimImage

```
Image Camera3P.aimImage
```

UI Image to display when aiming.

Definition at line 35 of file Camera3P.cs.

4.3.3.4 cam

```
Camera Camera3P.cam
```

Reference to the Camera component.

Definition at line 18 of file Camera3P.cs.

4.3.3.5 defaultCinemachine

```
CinemachineFreeLook Camera3P.defaultCinemachine
```

Default Cinemachine camera for third-person view.

Definition at line 41 of file Camera3P.cs.

4.3.3.6 disableRotation

```
bool Camera3P.disableRotation
```

Indicates if rotation is disabled.

Definition at line 38 of file Camera3P.cs.

4.3.3.7 horizontalI

```
float Camera3P.horizontalI  [private]
```

Horizontal input value.

Definition at line 28 of file Camera3P.cs.

4.3.3.8 lookDir

```
Transform Camera3P.lookDir
```

Direction the camera is looking at.

Definition at line 24 of file Camera3P.cs.

4.3.3.9 orientation

```
Transform Camera3P.orientation
```

Reference to the orientation transform.

Definition at line 12 of file Camera3P.cs.

4.3.3.10 player

```
Transform Camera3P.player
```

Reference to the player transform.

Definition at line 14 of file Camera3P.cs.

4.3.3.11 playerObject

```
Transform Camera3P.playerObject
```

Reference to the player object transform.

Definition at line 16 of file Camera3P.cs.

4.3.3.12 rotationSpeed

```
float Camera3P.rotationSpeed
```

Speed of camera rotation.

Definition at line 22 of file Camera3P.cs.

4.3.3.13 verticall

```
float Camera3P.verticalI  [private]
```

Vertical input value.

Definition at line 31 of file Camera3P.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Camera3P.cs

4.4 CheckPoint Class Reference

This class handles checkpoint functionality, saving the player's position when they reach a checkpoint.

Inherits MonoBehaviour.

Public Attributes

- checkPointsMenager menager
  Reference to the checkPointsManager that manages checkpoints.

Private Member Functions

- void OnTriggerEnter (Collider other)
  Called when another collider enters the checkpoint trigger.

4.4.1 Detailed Description

This class handles checkpoint functionality, saving the player's position when they reach a checkpoint.

Definition at line 6 of file CheckPoint.cs.

4.4.2 Member Function Documentation

4.4.2.1 OnTriggerEnter()

```
void CheckPoint.OnTriggerEnter (
            Collider other)  [private]
```

Called when another collider enters the checkpoint trigger.

Parameters

other	The Collider that triggered the checkpoint.

Definition at line 15 of file CheckPoint.cs.

4.4.3 Member Data Documentation

4.4.3.1 menager

```
checkPointsMenager CheckPoint.menager
```

Reference to the checkPointsManager that manages checkpoints.

Definition at line 9 of file CheckPoint.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/CheckPoint.cs

4.5 checkPointsMenager Class Reference

This class manages checkpoints and handles the player's position after death.

Inherits MonoBehaviour.

Public Member Functions

- Vector3 getPosition ()
  Gets the current respawn position.
- void setPosition (Vector3 position)
  Sets a new respawn position.

Public Attributes

- Transform Player
  Reference to the player's Transform.
- bool trigger = false
  Indicates if a checkpoint has been triggered.

Private Member Functions

- void Start ()
  Initializes the AfterDeadPosition to the player's initial position.
- void Update ()
  Updates the manager each frame.

Private Attributes

- Vector3 AfterDeadPosition
  Position to respawn the player after death.

4.5.1 Detailed Description

This class manages checkpoints and handles the player's position after death.

Definition at line 6 of file checkPointsMenager.cs.

4.5.2 Member Function Documentation

4.5.2.1 getPosition()

```
Vector3 checkPointsMenager.getPosition ()
```

Gets the current respawn position.

Returns

The position to respawn the player after death.

Definition at line 27 of file checkPointsMenager.cs.

4.5.2.2 setPosition()

```
void checkPointsMenager.setPosition (
            Vector3 position)
```

Sets a new respawn position.

Parameters

position	The new position to respawn the player.

Definition at line 37 of file checkPointsMenager.cs.

4.5.2.3 Start()

```
void checkPointsMenager.Start ()  [private]
```

Initializes the AfterDeadPosition to the player's initial position.

Definition at line 18 of file checkPointsMenager.cs.

4.5.2.4 Update()

```
void checkPointsMenager.Update ()  [private]
```

Updates the manager each frame.

Definition at line 45 of file checkPointsMenager.cs.

4.5.3 Member Data Documentation

4.5.3.1 AfterDeadPosition

```
Vector3 checkPointsMenager.AfterDeadPosition  [private]
```

Position to respawn the player after death.

Definition at line 9 of file checkPointsMenager.cs.

4.5.3.2 Player

```
Transform checkPointsMenager.Player
```

Reference to the player's Transform.

Definition at line 11 of file checkPointsMenager.cs.

4.5.3.3 trigger

```
bool checkPointsMenager.trigger = false
```

Indicates if a checkpoint has been triggered.

Definition at line 13 of file checkPointsMenager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/checkPointsMenager.cs

4.6 Climbing Class Reference

Handles the climbing mechanics for the player.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  Set properties values at the start of the game.
- void Update ()

  Set climbing state and handle climbing mechanics.
- void DisableClimbing ()

  Disables climbing.
- void Climb ()

  Handles climbing mechanics.
- void SetClimbingState ()

  Sets the climbing state based on various conditions.
- void SetClimbingLock ()

  Resets the climbing lock based on various conditions.

Private Attributes

- Transform orientation

  `orientation` *player's orientation object.*
- LayerMask climbableWall

  `climbableWall` *layer of climbable walls.*
- Camera3P camera3P

  `camera3P` *is reference to the Camera3P component, which is used to block the player's rotation when climbing.*
- float climbSpeed = 1.5f

  `climbSpeed` *the speed of climbing.*
- float maxClimbAngle = 30.0f

  `maxClimbAngle` *maximum angle to recognize climbing and walking on the wall.*
- float originalDrag

  `originalDrag` *original drag before start climbing.*
- bool isClimbing = false

  `isClimbing` *the state of climbing.*
- float maxClimbTimeX = 0.4f

  `maxClimbTimeX` *the maximum climbing times in the X direction.*
- float maxClimbTimeZ = 0.4f

  `maxClimbTimeZ` *the maximum climbing times in the Z direction.*
- RaycastHit hitWall

  `hitWall` *RaycastHit to store information about the wall on which the player is climbing.*
- readonly string[ ] climbingButtonsPrompts

  `climbingButtonsPrompts` *text for climbing button prompts.*
- ScreenHints buttonPromptsController

  `buttonPromptsController` *controller for button prompts on the screen.*
- float climbTimeX = 0.0f

  `climbTimeX` *and* `climbTimeZ` *current climbing time in X and Z directions.*

- float climbTimeZ = 0.0f
- PlayerMovement playerMovement

  `playerMovement` *reference to PlayerMovement to access touchGround variable.*
- bool climbingLock = false

  `climbingLock` *lock for climbing.*
- Rigidbody playerRigidbody

  `playerRigidbody` *the Rigidbody of the player.*
- Animator animator

  `animator` *the Animator of the player.*

4.6.1 Detailed Description

Handles the climbing mechanics for the player.

Definition at line 7 of file Climbing.cs.

4.6.2 Member Function Documentation

4.6.2.1 Climb()

```
void Climbing.Climb ()  [private]
```

Handles climbing mechanics.

Definition at line 124 of file Climbing.cs.

4.6.2.2 DisableClimbing()

```
void Climbing.DisableClimbing ()  [private]
```

Disables climbing.

Definition at line 101 of file Climbing.cs.

4.6.2.3 SetClimbingLock()

```
void Climbing.SetClimbingLock ()  [private]
```

Resets the climbing lock based on various conditions.

Definition at line 250 of file Climbing.cs.

4.6.2.4 SetClimbingState()

```
void Climbing.SetClimbingState ()  [private]
```

Sets the climbing state based on various conditions.

Definition at line 218 of file Climbing.cs.

4.6.2.5 Start()

```
void Climbing.Start ()  [private]
```

Set properties values at the start of the game.

Definition at line 74 of file Climbing.cs.

4.6.2.6 Update()

```
void Climbing.Update ()  [private]
```

Set climbing state and handle climbing mechanics.

Definition at line 85 of file Climbing.cs.

4.6.3 Member Data Documentation

4.6.3.1 animator

```
Animator Climbing.animator  [private]
```

`animator` the Animator of the player.

Definition at line 69 of file Climbing.cs.

4.6.3.2 buttonPromptsController

```
ScreenHints Climbing.buttonPromptsController  [private]
```

`buttonPromptsController` controller for button prompts on the screen.

Definition at line 53 of file Climbing.cs.

4.6.3.3 camera3P

```
Camera3P Climbing.camera3P  [private]
```

`camera3P` is reference to the Camera3P component, which is used to block the player's rotation when climbing.

Definition at line 19 of file Climbing.cs.

4.6.3.4 climbableWall

```
LayerMask Climbing.climbableWall  [private]
```

`climbableWall` layer of climbable walls.

Definition at line 15 of file Climbing.cs.

4.6.3.5 climbingButtonsPrompts

```
readonly string [] Climbing.climbingButtonsPrompts  [private]
```

Initial value:

```
=
    {
        "press <sprite name=\"E\"> to let go",
        "move with <sprite name=\"W\"> <sprite name=\"A\"> <sprite name=\"S\"> <sprite name=\"D\">"
    }
```

`climbingButtonsPrompts` text for climbing button prompts.

Definition at line 46 of file Climbing.cs.

4.6.3.6 climbingLock

```
bool Climbing.climbingLock = false  [private]
```

`climbingLock` lock for climbing.

Definition at line 63 of file Climbing.cs.

4.6.3.7 climbSpeed

```
float Climbing.climbSpeed = 1.5f  [private]
```

`climbSpeed` the speed of climbing.

Definition at line 24 of file Climbing.cs.

4.6.3.8 climbTimeX

```
float Climbing.climbTimeX = 0.0f  [private]
```

`climbTimeX` and `climbTimeZ` current climbing time in X and Z directions.

Definition at line 56 of file Climbing.cs.

4.6.3.9 climbTimeZ

```
float Climbing.climbTimeZ = 0.0f  [private]
```

Definition at line 57 of file Climbing.cs.

4.6.3.10 hitWall

```
RaycastHit Climbing.hitWall  [private]
```

`hitWall` RaycastHit to store information about the wall on which the player is climbing.

Definition at line 43 of file Climbing.cs.

4.6.3.11 isClimbing

```
bool Climbing.isClimbing = false  [private]
```

`isClimbing` the state of climbing.

Definition at line 32 of file Climbing.cs.

4.6.3.12 maxClimbAngle

```
float Climbing.maxClimbAngle = 30.0f  [private]
```

`maxClimbAngle` maximum angle to recognize climbing and walking on the wall.

Definition at line 27 of file Climbing.cs.

4.6.3.13 maxClimbTimeX

```
float Climbing.maxClimbTimeX = 0.4f  [private]
```

`maxClimbTimeX` the maximum climbing times in the X direction.

Definition at line 37 of file Climbing.cs.

4.6.3.14 maxClimbTimeZ

```
float Climbing.maxClimbTimeZ = 0.4f  [private]
```

`maxClimbTimeZ` the maximum climbing times in the Z direction.

Definition at line 40 of file Climbing.cs.

4.6.3.15 orientation

```
Transform Climbing.orientation  [private]
```

`orientation` player's orientation object.

Definition at line 12 of file Climbing.cs.

4.6.3.16 originalDrag

```
float Climbing.originalDrag  [private]
```

`originalDrag` original drag before start climbing.

Definition at line 30 of file Climbing.cs.

4.6.3.17 playerMovement

```
PlayerMovement Climbing.playerMovement  [private]
```

`playerMovement` reference to PlayerMovement to access touchGround variable.

Definition at line 60 of file Climbing.cs.

4.6.3.18 playerRigidbody

```
Rigidbody Climbing.playerRigidbody  [private]
```

`playerRigidbody` the Rigidbody of the player.

Definition at line 66 of file Climbing.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Climbing.cs

4.7 Collectible Class Reference

This class represents counting the number of collectibles the player has collected.

Inherits MonoBehaviour.

Public Attributes

- int collectedCounter = 0

  `collectedCounter` the number of collectibles the player has collected.

Private Member Functions

- void Start ()

  This method sets the levelStatistics reference when the game starts.
- void OnTriggerEnter (Collider other)

  This method is called when the Collider other enters the trigger. If the other object is a collectible, it is destroyed, and, CollectedCounter is incremented, and the collectedCount in LevelStatistics is updated.

Private Attributes

- LevelStatistics levelStatistics

  `levelStatistics` reference to LevelStatistics to set collectedCount to show.

4.7.1 Detailed Description

This class represents counting the number of collectibles the player has collected.

Definition at line 6 of file Collectible.cs.

4.7.2 Member Function Documentation

4.7.2.1 OnTriggerEnter()

```
void Collectible.OnTriggerEnter (
            Collider other)  [private]
```

This method is called when the Collider other enters the trigger. If the other object is a collectible, it is destroyed, and, CollectedCounter is incremented, and the collectedCount in LevelStatistics is updated.

Parameters

other	Object that player collides with.

Definition at line 28 of file Collectible.cs.

4.7.2.2 Start()

```
void Collectible.Start ()  [private]
```

This method sets the levelStatistics reference when the game starts.

Definition at line 17 of file Collectible.cs.

4.7.3 Member Data Documentation

4.7.3.1 collectedCounter

```
int Collectible.collectedCounter = 0
```

`collectedCounter` the number of collectibles the player has collected.

Definition at line 9 of file Collectible.cs.

4.7.3.2 levelStatistics

```
LevelStatistics Collectible.levelStatistics  [private]
```

`levelStatistics` reference to LevelStatistics to set collectedCount to show.

Definition at line 12 of file Collectible.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Collectible.cs

4.8 ColliderFromMesh Class Reference

This class creates a MeshCollider from a SkinnedMeshRenderer and updates it each frame.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()
  Initializes the components and sets up the MeshCollider.
- void Update ()
  Updates the MeshCollider with the current baked mesh each frame.

Private Attributes

- MeshCollider meshCollider

  Reference to the MeshCollider component.
- SkinnedMeshRenderer skinnedMeshRenderer

  Reference to the SkinnedMeshRenderer component.
- Mesh colliderMesh

  Mesh used for the collider.

4.8.1 Detailed Description

This class creates a MeshCollider from a SkinnedMeshRenderer and updates it each frame.

Definition at line 6 of file ColliderFromMesh.cs.

4.8.2 Member Function Documentation

4.8.2.1 Start()

```
void ColliderFromMesh.Start ()  [private]
```

Initializes the components and sets up the MeshCollider.

Definition at line 18 of file ColliderFromMesh.cs.

4.8.2.2 Update()

```
void ColliderFromMesh.Update ()  [private]
```

Updates the MeshCollider with the current baked mesh each frame.

Definition at line 38 of file ColliderFromMesh.cs.

4.8.3 Member Data Documentation

4.8.3.1 colliderMesh

```
Mesh ColliderFromMesh.colliderMesh  [private]
```

Mesh used for the collider.

Definition at line 13 of file ColliderFromMesh.cs.

4.8.3.2 meshCollider

```
MeshCollider ColliderFromMesh.meshCollider  [private]
```

Reference to the MeshCollider component.

Definition at line 9 of file ColliderFromMesh.cs.

4.8.3.3 skinnedMeshRenderer

```
SkinnedMeshRenderer ColliderFromMesh.skinnedMeshRenderer  [private]
```

Reference to the SkinnedMeshRenderer component.

Definition at line 11 of file ColliderFromMesh.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ColliderFromMesh.cs

4.9 Dash Class Reference

This class handles the dash ability for the player, including cooldown management and movement limits.

Inherits MonoBehaviour.

Public Attributes

- Transform cam

  Reference to the camera transform.
- GameObject playerObject

  Reference to the player object.
- float fullCooldown = 1.5f

  Full cooldown duration for the dash ability.
- float activeCooldown = 0f

  Current active cooldown time.
- float dashForce = 800f

  Force applied during the dash.
- float dashLimit = 80f

  Speed limit during the dash.
- float standardLimit = 25f

  Standard speed limit.

Private Member Functions

- void Start ()

  Initializes the Rigidbody and PlayerMovement components.
- void Update ()

  Updates the dash cooldown and checks for dash input each frame.
- void DashAbility ()

  Activates the dash ability, applying force and setting movement limits.
- void resetLimit ()

  Resets the player's movement speed limit to the standard limit.

Private Attributes

- PlayerMovement movement

  Reference to the PlayerMovement script.
- Rigidbody rb

  Reference to the Rigidbody component.

4.9.1 Detailed Description

This class handles the dash ability for the player, including cooldown management and movement limits.

Definition at line 6 of file Dash.cs.

4.9.2 Member Function Documentation

4.9.2.1 DashAbility()

```
void Dash.DashAbility ()  [private]
```

Activates the dash ability, applying force and setting movement limits.

Definition at line 60 of file Dash.cs.

4.9.2.2 resetLimit()

```
void Dash.resetLimit ()  [private]
```

Resets the player's movement speed limit to the standard limit.

Definition at line 77 of file Dash.cs.

4.9.2.3 Start()

```
void Dash.Start ()  [private]
```

Initializes the Rigidbody and PlayerMovement components.

Definition at line 33 of file Dash.cs.

4.9.2.4 Update()

```
void Dash.Update ()  [private]
```

Updates the dash cooldown and checks for dash input each frame.

Definition at line 42 of file Dash.cs.

4.9.3 Member Data Documentation

4.9.3.1 activeCooldown

```
float Dash.activeCooldown = 0f
```

Current active cooldown time.

Definition at line 22 of file Dash.cs.

4.9.3.2 cam

```
Transform Dash.cam
```

Reference to the camera transform.

Definition at line 12 of file Dash.cs.

4.9.3.3 dashForce

```
float Dash.dashForce = 800f
```

Force applied during the dash.

Definition at line 24 of file Dash.cs.

4.9.3.4 dashLimit

```
float Dash.dashLimit = 80f
```

Speed limit during the dash.

Definition at line 26 of file Dash.cs.

4.9.3.5 fullCooldown

```
float Dash.fullCooldown = 1.5f
```

Full cooldown duration for the dash ability.

Definition at line 20 of file Dash.cs.

4.9.3.6 movement

```
PlayerMovement Dash.movement  [private]
```

Reference to the PlayerMovement script.

Definition at line 10 of file Dash.cs.

4.9.3.7 playerObject

```
GameObject Dash.playerObject
```

Reference to the player object.

Definition at line 16 of file Dash.cs.

4.9.3.8 rb

```
Rigidbody Dash.rb  [private]
```

Reference to the Rigidbody component.

Definition at line 14 of file Dash.cs.

4.9.3.9 standardLimit

```
float Dash.standardLimit = 25f
```

Standard speed limit.

Definition at line 28 of file Dash.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Dash.cs

4.10 EnemyMovement Class Reference

This class handles enemy movement and behavior, including interaction with the player and responding to being kicked.

Inherits MonoBehaviour.

Public Member Functions

- void GetKicked ()

  Method to be called when the enemy gets kicked, stopping its movement.

Public Attributes

- NavMeshAgent agent

  Reference to the NavMeshAgent component for navigation.
- Transform playerTransform

  Reference to the player's transform.
- Animator animator

  Reference to the Animator component for handling animations.

Private Member Functions

- void Update ()

  Updates the enemy's behavior and animations each frame.
- void OnTriggerStay (Collider other)

  Handles the behavior when the player stays within the enemy's trigger range.
- void OnTriggerExit (Collider other)

  Handles the behavior when the player exits the enemy's trigger range.

Private Attributes

- bool playerInRange

  Indicates if the player is in range of the enemy.
- bool fighting

  Indicates if the enemy is fighting the player.
- bool stand

  Indicates if the enemy is standing.
- bool isKicked

  Indicates if the enemy has been kicked.

4.10.1 Detailed Description

This class handles enemy movement and behavior, including interaction with the player and responding to being kicked.

Definition at line 7 of file EnemyMovement.cs.

4.10.2 Member Function Documentation

4.10.2.1 GetKicked()

```
void EnemyMovement.GetKicked ()
```

Method to be called when the enemy gets kicked, stopping its movement.

Definition at line 105 of file EnemyMovement.cs.

4.10.2.2 OnTriggerExit()

```
void EnemyMovement.OnTriggerExit (
            Collider other)  [private]
```

Handles the behavior when the player exits the enemy's trigger range.

Parameters

other	The Collider that triggered the event.

Definition at line 97 of file EnemyMovement.cs.

4.10.2.3 OnTriggerStay()

```
void EnemyMovement.OnTriggerStay (
            Collider other)  [private]
```

Handles the behavior when the player stays within the enemy's trigger range.

Parameters

other	The Collider that triggered the event.

Definition at line 60 of file EnemyMovement.cs.

4.10.2.4 Update()

```
void EnemyMovement.Update ()  [private]
```

Updates the enemy's behavior and animations each frame.

Definition at line 29 of file EnemyMovement.cs.

4.10.3 Member Data Documentation

4.10.3.1 agent

```
NavMeshAgent EnemyMovement.agent
```

Reference to the NavMeshAgent component for navigation.

Definition at line 10 of file EnemyMovement.cs.

4.10.3.2 animator

```
Animator EnemyMovement.animator
```

Reference to the Animator component for handling animations.

Definition at line 24 of file EnemyMovement.cs.

4.10.3.3 fighting

```
bool EnemyMovement.fighting  [private]
```

Indicates if the enemy is fighting the player.

Definition at line 14 of file EnemyMovement.cs.

4.10.3.4 isKicked

```
bool EnemyMovement.isKicked  [private]
```

Indicates if the enemy has been kicked.

Definition at line 21 of file EnemyMovement.cs.

4.10.3.5 playerInRange

```
bool EnemyMovement.playerInRange  [private]
```

Indicates if the player is in range of the enemy.

Definition at line 12 of file EnemyMovement.cs.

4.10.3.6 playerTransform

```
Transform EnemyMovement.playerTransform
```

Reference to the player's transform.

Definition at line 16 of file EnemyMovement.cs.

4.10.3.7 stand

```
bool EnemyMovement.stand  [private]
```

Indicates if the enemy is standing.

Definition at line 18 of file EnemyMovement.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/EnemyMovement.cs

4.11 FinishLevelBarrel Class Reference

This class handles the finish level logic when a barrel enters the trigger.

Inherits MonoBehaviour.

Private Member Functions

- void OnTriggerEnter (Collider collider)
  Handles the behavior when another collider enters the trigger.

4.11.1 Detailed Description

This class handles the finish level logic when a barrel enters the trigger.

Definition at line 8 of file FinishLevelBarrel.cs.

4.11.2 Member Function Documentation

4.11.2.1 OnTriggerEnter()

```
void FinishLevelBarrel.OnTriggerEnter (
            Collider collider)  [private]
```

Handles the behavior when another collider enters the trigger.

Parameters

collider	The Collider that triggered the event.

Definition at line 14 of file FinishLevelBarrel.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/FinishLevelBarrel.cs

4.12 FPSTarget Class Reference

This class is used to set the target frame rate for the application.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  This method is called when the script instance is being loaded. It sets the vSyncCount to 0 and the target frame rate of the application.

Private Attributes

- int targetFrameRate = 144

  `targetFrameRate` *represents the target frame rate for the application.*

4.12.1 Detailed Description

This class is used to set the target frame rate for the application.

Definition at line 6 of file FPSTarget.cs.

4.12.2 Member Function Documentation

4.12.2.1 Start()

```
void FPSTarget.Start ()  [private]
```

This method is called when the script instance is being loaded. It sets the vSyncCount to 0 and the target frame rate of the application.

Definition at line 15 of file FPSTarget.cs.

4.12.3 Member Data Documentation

4.12.3.1 targetFrameRate

```
int FPSTarget.targetFrameRate = 144  [private]
```

`targetFrameRate` represents the target frame rate for the application.

Definition at line 9 of file FPSTarget.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/FPSTarget.cs

4.13 GoToLastCheckpoint Class Reference

This class handles resetting the player to the last checkpoint upon collision or trigger with specific objects.

Inherits MonoBehaviour.

Public Attributes

- checkPointsMenager menager
  Reference to the checkpoint manager.
- GameObject player
  Reference to the player game object.
- Animator animator
  Reference to the Animator component.

Private Member Functions

- void OnTriggerEnter (Collider other)
  Handles the behavior when another collider enters the trigger.
- void OnCollisionEnter (Collision collision)
  Handles the behavior when a collision occurs.

4.13.1 Detailed Description

This class handles resetting the player to the last checkpoint upon collision or trigger with specific objects.

Definition at line 6 of file GoToLastCheckpoint.cs.

4.13.2 Member Function Documentation

4.13.2.1 OnCollisionEnter()

```
void GoToLastCheckpoint.OnCollisionEnter (
            Collision collision)  [private]
```

Handles the behavior when a collision occurs.

Parameters

collision	The Collision that triggered the event.

Definition at line 34 of file GoToLastCheckpoint.cs.

4.13.2.2 OnTriggerEnter()

```
void GoToLastCheckpoint.OnTriggerEnter (
            Collider other)  [private]
```

Handles the behavior when another collider enters the trigger.

Parameters

other	The Collider that triggered the event.

Definition at line 19 of file GoToLastCheckpoint.cs.

4.13.3 Member Data Documentation

4.13.3.1 animator

```
Animator GoToLastCheckpoint.animator
```

Reference to the Animator component.

Definition at line 13 of file GoToLastCheckpoint.cs.

4.13.3.2 menager

```
checkPointsMenager GoToLastCheckpoint.menager
```

Reference to the checkpoint manager.

Definition at line 9 of file GoToLastCheckpoint.cs.

4.13.3.3 player

```
GameObject GoToLastCheckpoint.player
```

Reference to the player game object.

Definition at line 11 of file GoToLastCheckpoint.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/GoToLastCheckpoint.cs

4.14 GoToLastCheckpointOnMine Class Reference

This class manages respawning the player and a barrel at the last checkpoint upon collision with specific objects.

Inherits MonoBehaviour.

Public Attributes

- checkPointsMenager menager
  Reference to the checkpoint manager.
- GameObject player
  Reference to the player game object.
- GameObject barrel
  Reference to the barrel game object.
- PlayerMovement playerMovement
  Reference to the PlayerMovement script attached to the player.
- Animator animator
  Reference to the Animator component for handling animations.
- GameObject explosion
  Reference to the explosion game object for visual effects.
- bool exploding = false
  Indicates if the explosion animation is currently active.

Private Member Functions

- void Start ()
  Initializes the hash for the "isExploding" Animator parameter.
- void OnTriggerEnter (Collider other)
  Handles the behavior when another collider enters the trigger.
- void OnCollisionEnter (Collision collision)
  Handles the behavior when a collision occurs.
- IEnumerator RespawnPlayerWithDelay ()
  Coroutine to respawn the player with a delay after an explosion.
- IEnumerator RespawnBarrelWithDelay ()
  Coroutine to respawn the barrel with a delay after an explosion.

Private Attributes

- int isExplodingHash
  Hash for the "isExploding" parameter in the Animator component.

4.14.1 Detailed Description

This class manages respawning the player and a barrel at the last checkpoint upon collision with specific objects.

Definition at line 7 of file GoToLastCheckpointOnMine.cs.

4.14.2 Member Function Documentation

4.14.2.1 OnCollisionEnter()

```
void GoToLastCheckpointOnMine.OnCollisionEnter (
            Collision collision)  [private]
```

Handles the behavior when a collision occurs.

Parameters

collision	The Collision that triggered the event.

Definition at line 56 of file GoToLastCheckpointOnMine.cs.

4.14.2.2 OnTriggerEnter()

```
void GoToLastCheckpointOnMine.OnTriggerEnter (
            Collider other)  [private]
```

Handles the behavior when another collider enters the trigger.

Parameters

other	The Collider that triggered the event.

Definition at line 38 of file GoToLastCheckpointOnMine.cs.

4.14.2.3 RespawnBarrelWithDelay()

```
IEnumerator GoToLastCheckpointOnMine.RespawnBarrelWithDelay ()  [private]
```

Coroutine to respawn the barrel with a delay after an explosion.

Definition at line 97 of file GoToLastCheckpointOnMine.cs.

4.14.2.4 RespawnPlayerWithDelay()

```
IEnumerator GoToLastCheckpointOnMine.RespawnPlayerWithDelay ()  [private]
```

Coroutine to respawn the player with a delay after an explosion.

Definition at line 73 of file GoToLastCheckpointOnMine.cs.

4.14.2.5 Start()

```
void GoToLastCheckpointOnMine.Start ()  [private]
```

Initializes the hash for the "isExploding" Animator parameter.

Definition at line 29 of file GoToLastCheckpointOnMine.cs.

4.14.3 Member Data Documentation

4.14.3.1 animator

```
Animator GoToLastCheckpointOnMine.animator
```

Reference to the Animator component for handling animations.

Definition at line 18 of file GoToLastCheckpointOnMine.cs.

4.14.3.2 barrel

```
GameObject GoToLastCheckpointOnMine.barrel
```

Reference to the barrel game object.

Definition at line 14 of file GoToLastCheckpointOnMine.cs.

4.14.3.3 exploding

```
bool GoToLastCheckpointOnMine.exploding = false
```

Indicates if the explosion animation is currently active.

Definition at line 22 of file GoToLastCheckpointOnMine.cs.

4.14.3.4 explosion

```
GameObject GoToLastCheckpointOnMine.explosion
```

Reference to the explosion game object for visual effects.

Definition at line 20 of file GoToLastCheckpointOnMine.cs.

4.14.3.5 isExplodingHash

```
int GoToLastCheckpointOnMine.isExplodingHash  [private]
```

Hash for the "isExploding" parameter in the Animator component.

Definition at line 24 of file GoToLastCheckpointOnMine.cs.

4.14.3.6 menager

```
checkPointsMenager GoToLastCheckpointOnMine.menager
```

Reference to the checkpoint manager.

Definition at line 10 of file GoToLastCheckpointOnMine.cs.

4.14.3.7 player

```
GameObject GoToLastCheckpointOnMine.player
```

Reference to the player game object.

Definition at line 12 of file GoToLastCheckpointOnMine.cs.

4.14.3.8 playerMovement

```
PlayerMovement GoToLastCheckpointOnMine.playerMovement
```

Reference to the PlayerMovement script attached to the player.

Definition at line 16 of file GoToLastCheckpointOnMine.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/GoToLastCheckpointOnMine.cs

4.15 Grappling Class Reference

The Grappling class handles the grappling mechanics in the game.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  Initializes the grappling mechanics. Sets the line renderer and gun tip.
- void Update ()

  Handles the player's input for starting and stopping the grappling.
- void LateUpdate ()

  Draws the grappling line if the player is grappling.
- void DrawLine ()

  Draws the grappling line from the gun tip to the grapple point.
- void StartGrappling ()

  Starts the grappling if the player's aim hits a grappable object.
- void StopGrappling ()

  Stops the grappling and removes the grappling line.

Private Attributes

- LineRenderer lineRenderer

  `lineRenderer` *the line renderer for the grappling line.*
- Vector3 grapplePoint

  `grapplePoint` *the point where the grappling hook is attached.*
- Transform gunTip

  `gunTip` *the tip of the gun where the grappling hook is shot from.*
- bool grappling = false

  `grappling` *whether the player is currently grappling.*
- float maxGrapplingDistance

  `maxGrapplingDistance` *the maximum distance the grappling hook can reach.*
- Transform playerCamera

  `playerCamera` *the player's camera.*
- Transform playerObject

  `playerObject` *the player's object.*
- float spring = 10.0f

  `spring` *the spring value for the spring joint.*
- float damper = 7f

  `damper` *the damper value for the spring joint.*
- float massScale = 4.5f

  `massScale` *the mass scale value for the spring joint.*
- MultipleTags tags

  `tags` *the reference to the MultipleTags object.*
- string checkTag = "grappable"

  `checkTag` *the tag to check for grappable objects.*

4.15.1 Detailed Description

The Grappling class handles the grappling mechanics in the game.

Definition at line 7 of file Grappling.cs.

4.15.2 Member Function Documentation

4.15.2.1 DrawLine()

```
void Grappling.DrawLine ()  [private]
```

Draws the grappling line from the gun tip to the grapple point.

Definition at line 90 of file Grappling.cs.

4.15.2.2 LateUpdate()

```
void Grappling.LateUpdate ()  [private]
```

Draws the grappling line if the player is grappling.

Definition at line 79 of file Grappling.cs.

4.15.2.3 Start()

```
void Grappling.Start ()  [private]
```

Initializes the grappling mechanics. Sets the line renderer and gun tip.

Definition at line 54 of file Grappling.cs.

4.15.2.4 StartGrappling()

```
void Grappling.StartGrappling ()  [private]
```

Starts the grappling if the player's aim hits a grappable object.

Definition at line 99 of file Grappling.cs.

4.15.2.5 StopGrappling()

```
void Grappling.StopGrappling ()  [private]
```

Stops the grappling and removes the grappling line.

Definition at line 126 of file Grappling.cs.

4.15.2.6 Update()

```
void Grappling.Update ()  [private]
```

Handles the player's input for starting and stopping the grappling.

Definition at line 64 of file Grappling.cs.

4.15.3 Member Data Documentation

4.15.3.1 checkTag

```
string Grappling.checkTag = "grappable"  [private]
```

`checkTag` the tag to check for grappable objects.

Definition at line 49 of file Grappling.cs.

4.15.3.2 damper

```
float Grappling.damper = 7f  [private]
```

`damper` the damper value for the spring joint.

Definition at line 39 of file Grappling.cs.

4.15.3.3 grapplePoint

```
Vector3 Grappling.grapplePoint  [private]
```

`grapplePoint` the point where the grappling hook is attached.

Definition at line 13 of file Grappling.cs.

4.15.3.4 grappling

```
bool Grappling.grappling = false  [private]
```

`grappling` whether the player is currently grappling.

Definition at line 19 of file Grappling.cs.

4.15.3.5 gunTip

```
Transform Grappling.gunTip  [private]
```

`gunTip` the tip of the gun where the grappling hook is shot from.

Definition at line 16 of file Grappling.cs.

4.15.3.6 lineRenderer

```
LineRenderer Grappling.lineRenderer  [private]
```

`lineRenderer` the line renderer for the grappling line.

Definition at line 10 of file Grappling.cs.

4.15.3.7 massScale

```
float Grappling.massScale = 4.5f  [private]
```

`massScale` the mass scale value for the spring joint.

Definition at line 43 of file Grappling.cs.

4.15.3.8 maxGrapplingDistance

```
float Grappling.maxGrapplingDistance  [private]
```

`maxGrapplingDistance` the maximum distance the grappling hook can reach.

Definition at line 23 of file Grappling.cs.

4.15.3.9 playerCamera

```
Transform Grappling.playerCamera  [private]
```

`playerCamera` the player's camera.

Definition at line 27 of file Grappling.cs.

4.15.3.10 playerObject

```
Transform Grappling.playerObject  [private]
```

`playerObject` the player's object.

Definition at line 31 of file Grappling.cs.

4.15.3.11 spring

```
float Grappling.spring = 10.0f  [private]
```

`spring` the spring value for the spring joint.

Definition at line 35 of file Grappling.cs.

4.15.3.12 tags

```
MultipleTags Grappling.tags  [private]
```

`tags` the reference to the MultipleTags object.

Definition at line 46 of file Grappling.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Grappling.cs

4.16 InteractionWithObjects Class Reference

This class manages interactions with objects based on raycasting from the player's position.

Inherits MonoBehaviour.

Public Attributes

- GameObject Interactinfo

  Reference to the GameObject displaying interaction information.
- LayerMask interactable

  Layer mask for interactable objects.
- LayerMask defaultLayer

  Default layer mask.
- GameObject playerObject

  Reference to the player's GameObject.
- string interaction = "animate"

  Type of interaction ("animate" or "exit map").
- LevelStatistics ls

  Reference to the LevelStatistics script.
- int levelUnlocked

  Level to unlock upon interaction.
- float rayDistance

  Distance of the raycast for interaction.

Private Member Functions

- void Update ()

Private Attributes

- RaycastHit hit

4.16.1 Detailed Description

This class manages interactions with objects based on raycasting from the player's position.

Definition at line 7 of file InteractionWithObjects.cs.

4.16.2 Member Function Documentation

4.16.2.1 Update()

```
void InteractionWithObjects.Update ()  [private]
```

Definition at line 32 of file InteractionWithObjects.cs.

4.16.3 Member Data Documentation

4.16.3.1 defaultLayer

```
LayerMask InteractionWithObjects.defaultLayer
```

Default layer mask.

Definition at line 15 of file InteractionWithObjects.cs.

4.16.3.2 hit

```
RaycastHit InteractionWithObjects.hit  [private]
```

Definition at line 29 of file InteractionWithObjects.cs.

4.16.3.3 interactable

```
LayerMask InteractionWithObjects.interactable
```

Layer mask for interactable objects.

Definition at line 13 of file InteractionWithObjects.cs.

4.16.3.4 Interactinfo

```
GameObject InteractionWithObjects.Interactinfo
```

Reference to the GameObject displaying interaction information.

Definition at line 11 of file InteractionWithObjects.cs.

4.16.3.5 interaction

```
string InteractionWithObjects.interaction = "animate"
```

Type of interaction ("animate" or "exit map").

Definition at line 19 of file InteractionWithObjects.cs.

4.16.3.6 levelUnlocked

```
int InteractionWithObjects.levelUnlocked
```

Level to unlock upon interaction.

Definition at line 23 of file InteractionWithObjects.cs.

4.16.3.7 ls

```
LevelStatistics InteractionWithObjects.ls
```

Reference to the LevelStatistics script.

Definition at line 21 of file InteractionWithObjects.cs.

4.16.3.8 playerObject

```
GameObject InteractionWithObjects.playerObject
```

Reference to the player's GameObject.

Definition at line 17 of file InteractionWithObjects.cs.

4.16.3.9 rayDistance

```
float InteractionWithObjects.rayDistance
```

Distance of the raycast for interaction.

Definition at line 27 of file InteractionWithObjects.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/InteractionWithObjects.cs

4.17 kickEnemy Class Reference

This class manages kicking enemies when they collide with a trigger.

Inherits MonoBehaviour.

Public Attributes

- Animator playerAnimator
  Reference to the Animator of the player.

Private Member Functions

- void Start ()
- void OnTriggerEnter (Collider other)
  Handles the behavior when another collider enters the trigger.

Private Attributes

- bool getKicked
  Flag indicating if the enemy has been kicked

4.17.1 Detailed Description

This class manages kicking enemies when they collide with a trigger.

Definition at line 8 of file kickEnemy.cs.

4.17.2 Member Function Documentation

4.17.2.1 OnTriggerEnter()

```
void kickEnemy.OnTriggerEnter (
            Collider other)  [private]
```

Handles the behavior when another collider enters the trigger.

Parameters

other	The Collider that triggered the event.

Definition at line 25 of file kickEnemy.cs.

4.17.2.2 Start()

```
void kickEnemy.Start ()  [private]
```

Definition at line 16 of file kickEnemy.cs.

4.17.3 Member Data Documentation

4.17.3.1 getKicked

```
bool kickEnemy.getKicked  [private]
```

Flag indicating if the enemy has been kicked

Definition at line 13 of file kickEnemy.cs.

4.17.3.2 playerAnimator

```
Animator kickEnemy.playerAnimator
```

Reference to the Animator of the player.

Definition at line 11 of file kickEnemy.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/kickEnemy.cs

4.18 Level4WaterReset Class Reference

The Level4WaterReset class is responsible for resetting the player's position when the player is underwater. It checks if the player is underwater and if so, starts a countdown. If the player stays underwater for a certain amount of time, the player's position is reset.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  Initialization of player, playerHeight, and waterLevel variables.
- void Update ()

  Checks if the player has fallen into the water and if so, starts a countdown to reset the player's position.
- void ResetPlayerPosition ()

  The ResetPlayerPosition method resets the player's position to the startPoint and stops any movement.

Private Attributes

- double waterLevel

  `waterLevel` *is the y-coordinate of the water's surface.*
- double playerHeight

  `playerHeight` *is the height of the player.*
- TMP_Text underWaterText

  `underWaterText` *is the TextMeshPro component where the countdown is displayed.*
- GameObject player

  `player` *is the player's GameObject.*
- Transform startPoint

  `startPoint` *is the Transform where the player is reset to after falling into the water.*
- float numberOfSecondsToWait = 5

  `numberOfSecondsToWait` *is the number of seconds to wait before resetting the player's position.*
- float timeElapsed = 0

  `timeElapsed` *is the time in seconds that has elapsed since the player fell into the water.*
- int secondsElapsed = 0

  `secondsElapsed` *is the number of whole seconds that has elapsed since the player fell into the water.*

4.18.1 Detailed Description

The Level4WaterReset class is responsible for resetting the player's position when the player is underwater. It checks if the player is underwater and if so, starts a countdown. If the player stays underwater for a certain amount of time, the player's position is reset.

Definition at line 9 of file Level4WaterReset.cs.

4.18.2 Member Function Documentation

4.18.2.1 ResetPlayerPosition()

```
void Level4WaterReset.ResetPlayerPosition ()  [private]
```

The ResetPlayerPosition method resets the player's position to the startPoint and stops any movement.

Definition at line 75 of file Level4WaterReset.cs.

4.18.2.2 Start()

```
void Level4WaterReset.Start ()  [private]
```

Initialization of player, playerHeight, and waterLevel variables.

Definition at line 41 of file Level4WaterReset.cs.

4.18.2.3 Update()

```
void Level4WaterReset.Update ()  [private]
```

Checks if the player has fallen into the water and if so, starts a countdown to reset the player's position.

Definition at line 51 of file Level4WaterReset.cs.

4.18.3 Member Data Documentation

4.18.3.1 numberOfSecondsToWait

```
float Level4WaterReset.numberOfSecondsToWait = 5  [private]
```

`numberOfSecondsToWait` is the number of seconds to wait before resetting the player's position.

Definition at line 30 of file Level4WaterReset.cs.

4.18.3.2 player

```
GameObject Level4WaterReset.player  [private]
```

`player` is the player's GameObject.

Definition at line 22 of file Level4WaterReset.cs.

4.18.3.3 playerHeight

```
double Level4WaterReset.playerHeight  [private]
```

`playerHeight` is the height of the player.

Definition at line 15 of file Level4WaterReset.cs.

4.18.3.4 secondsElapsed

```
int Level4WaterReset.secondsElapsed = 0  [private]
```

`secondsElapsed` is the number of whole seconds that has elapsed since the player fell into the water.

Definition at line 36 of file Level4WaterReset.cs.

4.18.3.5 startPoint

```
Transform Level4WaterReset.startPoint  [private]
```

`startPoint` is the Transform where the player is reset to after falling into the water.

Definition at line 26 of file Level4WaterReset.cs.

4.18.3.6 timeElapsed

```
float Level4WaterReset.timeElapsed = 0  [private]
```

`timeElapsed` is the time in seconds that has elapsed since the player fell into the water.

Definition at line 33 of file Level4WaterReset.cs.

4.18.3.7 underWaterText

```
TMP_Text Level4WaterReset.underWaterText  [private]
```

`underWaterText` is the TextMeshPro component where the countdown is displayed.

Definition at line 19 of file Level4WaterReset.cs.

4.18.3.8 waterLevel

```
double Level4WaterReset.waterLevel  [private]
```

`waterLevel` is the y-coordinate of the water's surface.

Definition at line 12 of file Level4WaterReset.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Level4WaterReset.cs

4.19 Level5LavaReset Class Reference

The Level5LavaReset class is responsible for resetting the player's position when the player fell into the lava. It checks if the player fell into lava and if so, starts a countdown. After a certain amount of time, the player's position is reset.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  The Start method is called before the first frame update. It initializes the player, playerHeight, and lavaLevel variables.
- void Update ()

  The Update method is called once per frame. It checks if the player has fallen into the lava and if so, starts a countdown to reset the player's position.
- void ResetPlayerPosition ()

  The ResetPlayerPosition method resets the player's position to the startPoint and stops any movement.

Private Attributes

- double lavaLevel

  `lavaLevel` *is the y-coordinate of the lava's surface.*
- double playerHeight

  `playerHeight` *is the height of the player.*
- TMP_Text textGUI

  `textGUI` *is the TextMeshPro component where the countdown is displayed.*
- GameObject player

  `player` *is the player's GameObject.*
- Transform startPoint

  `startPoint` *is the Transform where the player is reset to after falling into the lava.*
- float numberOfSecondsToWait = 5

  `numberOfSecondsToWait` *is the number of seconds to wait before resetting the player's position.*
- GameObject lavaObject

  `lavaObject` *is the GameObject of the lava.*
- float timeElapsed = 0

  `timeElapsed` *is the time in seconds that has elapsed since the player fell into the lava.*
- int secondsElapsed = 0

  `secondsElapsed` *is the number of whole seconds that has elapsed since the player fell into the lava.*

4.19.1 Detailed Description

The Level5LavaReset class is responsible for resetting the player's position when the player fell into the lava. It checks if the player fell into lava and if so, starts a countdown. After a certain amount of time, the player's position is reset.

Definition at line 10 of file Level5LavaReset.cs.

4.19.2 Member Function Documentation

4.19.2.1 ResetPlayerPosition()

```
void Level5LavaReset.ResetPlayerPosition ()  [private]
```

The ResetPlayerPosition method resets the player's position to the startPoint and stops any movement.

Definition at line 81 of file Level5LavaReset.cs.

4.19.2.2 Start()

```
void Level5LavaReset.Start ()  [private]
```

The Start method is called before the first frame update. It initializes the player, playerHeight, and lavaLevel variables.

Definition at line 46 of file Level5LavaReset.cs.

4.19.2.3 Update()

```
void Level5LavaReset.Update ()  [private]
```

The Update method is called once per frame. It checks if the player has fallen into the lava and if so, starts a countdown to reset the player's position.

Definition at line 56 of file Level5LavaReset.cs.

4.19.3 Member Data Documentation

4.19.3.1 lavaLevel

```
double Level5LavaReset.lavaLevel  [private]
```

`lavaLevel` is the y-coordinate of the lava's surface.

Definition at line 13 of file Level5LavaReset.cs.

4.19.3.2 lavaObject

```
GameObject Level5LavaReset.lavaObject  [private]
```

`lavaObject` is the GameObject of the lava.

Definition at line 35 of file Level5LavaReset.cs.

4.19.3.3 numberOfSecondsToWait

```
float Level5LavaReset.numberOfSecondsToWait = 5  [private]
```

`numberOfSecondsToWait` is the number of seconds to wait before resetting the player's position.

Definition at line 31 of file Level5LavaReset.cs.

4.19.3.4 player

```
GameObject Level5LavaReset.player  [private]
```

`player` is the player's GameObject.

Definition at line 23 of file Level5LavaReset.cs.

4.19.3.5 playerHeight

```
double Level5LavaReset.playerHeight  [private]
```

`playerHeight` is the height of the player.

Definition at line 16 of file Level5LavaReset.cs.

4.19.3.6 secondsElapsed

```
int Level5LavaReset.secondsElapsed = 0  [private]
```

`secondsElapsed` is the number of whole seconds that has elapsed since the player fell into the lava.

Definition at line 41 of file Level5LavaReset.cs.

4.19.3.7 startPoint

```
Transform Level5LavaReset.startPoint  [private]
```

`startPoint` is the Transform where the player is reset to after falling into the lava.

Definition at line 27 of file Level5LavaReset.cs.

4.19.3.8 textGUI

```
TMP_Text Level5LavaReset.textGUI  [private]
```

`textGUI` is the TextMeshPro component where the countdown is displayed.

Definition at line 20 of file Level5LavaReset.cs.

4.19.3.9 timeElapsed

```
float Level5LavaReset.timeElapsed = 0  [private]
```

`timeElapsed` is the time in seconds that has elapsed since the player fell into the lava.

Definition at line 38 of file Level5LavaReset.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Level5LavaReset.cs

4.20 LevelStatistics Class Reference

The LevelStatistics class handles the tracking and display of collectible items in a level.

Inherits MonoBehaviour.

Public Attributes

- int collectedCount = 0

  `collectedCount` *is the count of collected items.*
- int totalCollectibleCount = 0

  `totalCollectibleCount` *is the total count of collectible items in the level.*

Private Member Functions

- void Start ()

  Initialization of the totalCollectibleCount.
- void LateUpdate ()

  The LateUpdate method is called once per frame, after all Update methods have been called. It updates the display message.

Private Attributes

- TMP_Text canvasText

  `canvasText` *is the TextMeshPro component where the statistics are displayed.*
- string[ ] messages

  `messages` *is an array of strings that form the display message.*

4.20.1 Detailed Description

The LevelStatistics class handles the tracking and display of collectible items in a level.

Definition at line 7 of file LevelStatistics.cs.

4.20.2 Member Function Documentation

4.20.2.1 LateUpdate()

```
void LevelStatistics.LateUpdate ()  [private]
```

The LateUpdate method is called once per frame, after all Update methods have been called. It updates the display message.

Definition at line 38 of file LevelStatistics.cs.

4.20.2.2 Start()

```
void LevelStatistics.Start ()  [private]
```

Initialization of the totalCollectibleCount.

Definition at line 30 of file LevelStatistics.cs.

4.20.3 Member Data Documentation

4.20.3.1 canvasText

```
TMP_Text LevelStatistics.canvasText  [private]
```

`canvasText` is the TextMeshPro component where the statistics are displayed.

Definition at line 18 of file LevelStatistics.cs.

4.20.3.2 collectedCount

```
int LevelStatistics.collectedCount = 0
```

`collectedCount` is the count of collected items.

Definition at line 10 of file LevelStatistics.cs.

4.20.3.3 messages

```
string [] LevelStatistics.messages  [private]
```

Initial value:

```
=
    {
        "Level Statistics",
        ""
    }
```

`messages` is an array of strings that form the display message.

Definition at line 21 of file LevelStatistics.cs.

4.20.3.4 totalCollectibleCount

```
int LevelStatistics.totalCollectibleCount = 0
```

`totalCollectibleCount` is the total count of collectible items in the level.

Definition at line 14 of file LevelStatistics.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/LevelStatistics.cs

4.21 Lift Class Reference

The Lift class handles the movement of the lift in the game.

Inherits MonoBehaviour.

Public Member Functions

- void ActivateLift ()

  Activates the lift, causing the platform to move up or down depending on its current position.
- void MoveLiftUp ()

  Moves the lift platform up.
- void MoveLiftDown ()

  Moves the lift platform down.

Private Member Functions

- void Start ()

  Initialization of the lift platform, chains, and positions at the start of the game.
- void Update ()

  Handles the movement of the lift platform and chains.
- void UpdateTextureOffset ()

  Updates the texture offset of the lift chains.

Private Attributes

- float speed = 0.008f

  The speed at which the lift moves.
- GameObject platformUpPost

  The upper position of the lift platform.
- GameObject platformDownPost

  The lower position of the lift platform.
- Renderer chain1Renderer

  The Renderer component of the first chain of the lift.
- Renderer chain2Renderer

  The Renderer component of the second chain of the lift.
- Rigidbody platform

  The Rigidbody component of the lift platform.
- bool isPlatformUp = true

  A flag indicating whether the lift platform is up.
- bool isPlatformMoving = false

  A flag indicating whether the lift platform is moving.

4.21.1 Detailed Description

The Lift class handles the movement of the lift in the game.

Definition at line 6 of file Lift.cs.

4.21.2 Member Function Documentation

4.21.2.1 ActivateLift()

```
void Lift.ActivateLift ()
```

Activates the lift, causing the platform to move up or down depending on its current position.

Definition at line 118 of file Lift.cs.

4.21.2.2 MoveLiftDown()

```
void Lift.MoveLiftDown ()
```

Moves the lift platform down.

Definition at line 140 of file Lift.cs.

4.21.2.3 MoveLiftUp()

```
void Lift.MoveLiftUp ()
```

Moves the lift platform up.

Definition at line 129 of file Lift.cs.

4.21.2.4 Start()

```
void Lift.Start ()  [private]
```

Initialization of the lift platform, chains, and positions at the start of the game.

Definition at line 52 of file Lift.cs.

4.21.2.5 Update()

```
void Lift.Update ()  [private]
```

Handles the movement of the lift platform and chains.

Definition at line 66 of file Lift.cs.

4.21.2.6 UpdateTextureOffset()

```
void Lift.UpdateTextureOffset ()  [private]
```

Updates the texture offset of the lift chains.

Definition at line 102 of file Lift.cs.

4.21.3 Member Data Documentation

4.21.3.1 chain1Renderer

```
Renderer Lift.chain1Renderer  [private]
```

The Renderer component of the first chain of the lift.

Definition at line 27 of file Lift.cs.

4.21.3.2 chain2Renderer

```
Renderer Lift.chain2Renderer  [private]
```

The Renderer component of the second chain of the lift.

Definition at line 32 of file Lift.cs.

4.21.3.3 isPlatformMoving

```
bool Lift.isPlatformMoving = false  [private]
```

A flag indicating whether the lift platform is moving.

Definition at line 47 of file Lift.cs.

4.21.3.4 isPlatformUp

```
bool Lift.isPlatformUp = true  [private]
```

A flag indicating whether the lift platform is up.

Definition at line 42 of file Lift.cs.

4.21.3.5 platform

```
Rigidbody Lift.platform  [private]
```

The Rigidbody component of the lift platform.

Definition at line 37 of file Lift.cs.

4.21.3.6 platformDownPost

```
GameObject Lift.platformDownPost  [private]
```

The lower position of the lift platform.

Definition at line 22 of file Lift.cs.

4.21.3.7 platformUpPost

```
GameObject Lift.platformUpPost  [private]
```

The upper position of the lift platform.

Definition at line 17 of file Lift.cs.

4.21.3.8 speed

```
float Lift.speed = 0.008f  [private]
```

The speed at which the lift moves.

Definition at line 12 of file Lift.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Lift.cs

4.22 LiftActivation Class Reference

The LiftActivation class handles the activation of lifts in the game when the player enters a specific trigger area.

Inherits MonoBehaviour.

Private Member Functions

- void OnTriggerEnter (Collider other)

  This method checks if the player has entered the trigger area and, if so, activates the lift. Depending on the name of the trigger area, the lift is moved up, down, or activated (moving between positions).

4.22.1 Detailed Description

The LiftActivation class handles the activation of lifts in the game when the player enters a specific trigger area.

Definition at line 6 of file LiftActivation.cs.

4.22.2 Member Function Documentation

4.22.2.1 OnTriggerEnter()

```
void LiftActivation.OnTriggerEnter (
            Collider other)  [private]
```

This method checks if the player has entered the trigger area and, if so, activates the lift. Depending on the name of the trigger area, the lift is moved up, down, or activated (moving between positions).

Parameters

other	The other Collider involved in this collision. (Player)

Definition at line 13 of file LiftActivation.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/LiftActivation.cs

4.23 LineMovement Class Reference

The LineMovement class handles the player's movement along movable lines in the game.

Inherits MonoBehaviour.

Public Member Functions

- bool IsMovingOnLine ()

  IsMovingOnLine checks if the player is currently moving on the line.

Public Attributes

- bool isMovingOnLine = false

  `isMovingOnLine` *is a flag indicating whether the player is currently moving on the line.*
- Camera3P camera3P

  `camera3P` *is reference to the Camera3P component, which is used to block the player's rotation when moving on the line.*
- int direction = 1

  `direction` *is the direction in which the player is moving along the line. It is used to move the player in current looking direction.*
- bool isAboveLine = false

  `isAboveLine` *is a flag indicating whether the player is currently above the line.*

Private Member Functions

- void calculateLineDirection (Collision collision)

  Calculates the direction of the line based on the collision.
- void OnCollisionEnter (Collision collision)

  It handles the player's interaction with the movable line. Sets the player's movement along the line and displays the button prompts to the player.
- void OnCollisionExit (Collision collision)

  It is called when player is no longer in contact with the movable line. It handles the player's disengagement from the movable line.
- void Start ()

  It initializes the player's Rigidbody, ScreenHints, and PlayerMovement components at the start of the game.
- void Update ()

  It handles the player's movement along the movable line and disengagement from the movable line.

Private Attributes

- float moveTime = 0.0f

  moveTime is the timer for the player's movement along the line after releasing the movement keys.
- float moveTimeMax = 0.1f

  moveTimeMax is the maximum time for the player's movement along the line after releasing the movement keys.
- float speed = 3.0f

  speed is the speed at which the player moves along the line.
- Transform playerOrientation

  playerOrientation is the player's orientation in the game world.
- PlayerMovement playerMovement

  playerMovement is the player's movement component, which is used to disable and enable the player's air movement.
- Vector3 lineDirection

  lineDirection is the direction of the line.
- ScreenHints buttonPromptsController

  buttonPromptsController is the controller for the button prompts displayed to the player.
- readonly string[ ] movingAboveLineButtonPrompts

  movingAboveLineButtonPrompts is the button prompt displayed to the player when he is moving above the line.
- readonly string[ ] movingUnderLineButtonPrompts

  movingUnderLineButtonPrompts is the button prompt displayed to the player when he is moving under the line.
- Rigidbody playerRigidbody

  playerRigidbody is the player's Rigidbody component.
- Quaternion initialPlayerRotation

  initialPlayerRotation is the initial rotation of the player when moving under the line. It is used to set minimap icon rotation properly.
- Transform minimapIcon

  minimapIcon is the minimap icon of the player.
- int minimapRotChangeY = 90

  minimapRotChangeY is the rotation change in the y-axis for the minimap icon when moving under the line.

4.23.1 Detailed Description

The LineMovement class handles the player's movement along movable lines in the game.

Definition at line 7 of file LineMovement.cs.

4.23.2 Member Function Documentation

4.23.2.1 calculateLineDirection()

```
void LineMovement.calculateLineDirection (
            Collision collision)  [private]
```

Calculates the direction of the line based on the collision.

Definition at line 75 of file LineMovement.cs.

4.23.2.2 IsMovingOnLine()

```
bool LineMovement.IsMovingOnLine ()
```

IsMovingOnLine checks if the player is currently moving on the line.

Definition at line 242 of file LineMovement.cs.

4.23.2.3 OnCollisionEnter()

```
void LineMovement.OnCollisionEnter (
            Collision collision)  [private]
```

It handles the player's interaction with the movable line. Sets the player's movement along the line and displays the button prompts to the player.

Definition at line 111 of file LineMovement.cs.

4.23.2.4 OnCollisionExit()

```
void LineMovement.OnCollisionExit (
            Collision collision)  [private]
```

It is called when player is no longer in contact with the movable line. It handles the player's disengagement from the movable line.

Definition at line 157 of file LineMovement.cs.

4.23.2.5 Start()

```
void LineMovement.Start ()  [private]
```

It initializes the player's Rigidbody, ScreenHints, and PlayerMovement components at the start of the game.

Definition at line 177 of file LineMovement.cs.

4.23.2.6 Update()

```
void LineMovement.Update ()  [private]
```

It handles the player's movement along the movable line and disengagement from the movable line.

Definition at line 188 of file LineMovement.cs.

4.23.3 Member Data Documentation

4.23.3.1 buttonPromptsController

```
ScreenHints LineMovement.buttonPromptsController  [private]
```

`buttonPromptsController` is the controller for the button prompts displayed to the player.

Definition at line 39 of file LineMovement.cs.

4.23.3.2 camera3P

```
Camera3P LineMovement.camera3P
```

`camera3P` is reference to the Camera3P component, which is used to block the player's rotation when moving on the line.

Definition at line 30 of file LineMovement.cs.

4.23.3.3 direction

```
int LineMovement.direction = 1
```

`direction` is the direction in which the player is moving along the line. It is used to move the player in current looking direction.

Definition at line 58 of file LineMovement.cs.

4.23.3.4 initialPlayerRotation

```
Quaternion LineMovement.initialPlayerRotation  [private]
```

`initialPlayerRotation` is the initial rotation of the player when moving under the line. It is used to set minimap icon rotation properly.

Definition at line 64 of file LineMovement.cs.

4.23.3.5 isAboveLine

```
bool LineMovement.isAboveLine = false
```

`isAboveLine` is a flag indicating whether the player is currently above the line.

Definition at line 61 of file LineMovement.cs.

4.23.3.6 isMovingOnLine

```
bool LineMovement.isMovingOnLine = false
```

`isMovingOnLine` is a flag indicating whether the player is currently moving on the line.

Definition at line 21 of file LineMovement.cs.

4.23.3.7 lineDirection

```
Vector3 LineMovement.lineDirection  [private]
```

`lineDirection` is the direction of the line.

Definition at line 36 of file LineMovement.cs.

4.23.3.8 minimapIcon

```
Transform LineMovement.minimapIcon  [private]
```

minimapIcon is the minimap icon of the player.

Definition at line 67 of file LineMovement.cs.

4.23.3.9 minimapRotChangeY

```
int LineMovement.minimapRotChangeY = 90  [private]
```

`minimapRotChangeY` is the rotation change in the y-axis for the minimap icon when moving under the line.

Definition at line 70 of file LineMovement.cs.

4.23.3.10 moveTime

```
float LineMovement.moveTime = 0.0f  [private]
```

`moveTime` is the timer for the player's movement along the line after releasing the movement keys.

Definition at line 11 of file LineMovement.cs.

4.23.3.11 moveTimeMax

```
float LineMovement.moveTimeMax = 0.1f  [private]
```

`moveTimeMax` is the maximum time for the player's movement along the line after releasing the movement keys.

Definition at line 13 of file LineMovement.cs.

4.23.3.12 movingAboveLineButtonPrompts

```
readonly string [] LineMovement.movingAboveLineButtonPrompts  [private]
```

Initial value:

```
=
    {
        "press <sprite name=\"W\"> and <sprite name=\"S\"> to move forward and backward"
    }
```

`movingAboveLineButtonPrompts` is the button prompt displayed to the player when he is moving above the line.

Definition at line 42 of file LineMovement.cs.

4.23.3.13 movingUnderLineButtonPrompts

```
readonly string [] LineMovement.movingUnderLineButtonPrompts  [private]
```

Initial value:

```
=
    {
        "press <sprite name=\"E\"> to let go",
        "press <sprite name=\"W\"> and <sprite name=\"S\"> to move forward and backward"
    }
```

`movingUnderLineButtonPrompts` is the button prompt displayed to the player when he is moving under the line.

Definition at line 48 of file LineMovement.cs.

4.23.3.14 playerMovement

```
PlayerMovement LineMovement.playerMovement  [private]
```

`playerMovement` is the player's movement component, which is used to disable and enable the player's air movement.

Definition at line 33 of file LineMovement.cs.

4.23.3.15 playerOrientation

```
Transform LineMovement.playerOrientation  [private]
```

`playerOrientation` is the player's orientation in the game world.

Definition at line 26 of file LineMovement.cs.

4.23.3.16 playerRigidbody

```
Rigidbody LineMovement.playerRigidbody  [private]
```

`playerRigidbody` is the player's Rigidbody component.

Definition at line 55 of file LineMovement.cs.

4.23.3.17 speed

```
float LineMovement.speed = 3.0f  [private]
```

`speed` is the speed at which the player moves along the line.

Definition at line 18 of file LineMovement.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/LineMovement.cs

4.24 MainMenu Class Reference

This class manages the main menu interactions such as starting the game and quitting the application.

Inherits MonoBehaviour.

Public Member Functions

- void PlayGame ()
  Loads the main game scene.
- void QuitGame ()
  Quits the application.

4.24.1 Detailed Description

This class manages the main menu interactions such as starting the game and quitting the application.

Definition at line 9 of file MainMenu.cs.

4.24.2 Member Function Documentation

4.24.2.1 PlayGame()

```
void MainMenu.PlayGame ()
```

Loads the main game scene.

Definition at line 14 of file MainMenu.cs.

4.24.2.2 QuitGame()

```
void MainMenu.QuitGame ()
```

Quits the application.

Definition at line 22 of file MainMenu.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/MainMenu.cs

4.25 Manager Class Reference

This class serves as a manager for general game functionalities.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  Called before the first frame update. Uncommenting SaveSystem.Reset() would reset any necessary game state upon startup.
- void Update ()

  Called once per frame.

4.25.1 Detailed Description

This class serves as a manager for general game functionalities.

Definition at line 6 of file Manager.cs.

4.25.2 Member Function Documentation

4.25.2.1 Start()

```
void Manager.Start ()  [private]
```

Called before the first frame update. Uncommenting SaveSystem.Reset() would reset any necessary game state upon startup.

Definition at line 11 of file Manager.cs.

4.25.2.2 Update()

```
void Manager.Update ()  [private]
```

Called once per frame.

Definition at line 19 of file Manager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Manager.cs

4.26 MiniMapCamera Class Reference

The MiniMapCamera class handles the positioning and rotation of the minimap camera.

Inherits MonoBehaviour.

Private Member Functions

- void LateUpdate ()

  Updating of the position and rotation of the minimap camera to follow the player.

Private Attributes

- Transform player

  `player` *is the Transform of the player object that the minimap camera follows.*

4.26.1 Detailed Description

The MiniMapCamera class handles the positioning and rotation of the minimap camera.

Definition at line 6 of file MiniMapCamera.cs.

4.26.2 Member Function Documentation

4.26.2.1 LateUpdate()

```
void MiniMapCamera.LateUpdate ()  [private]
```

Updating of the position and rotation of the minimap camera to follow the player.

Definition at line 15 of file MiniMapCamera.cs.

4.26.3 Member Data Documentation

4.26.3.1 player

```
Transform MiniMapCamera.player  [private]
```

`player` is the Transform of the player object that the minimap camera follows.

Definition at line 10 of file MiniMapCamera.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/MiniMapCamera.cs

4.27 MovableCameraController Class Reference

The MovableCameraController class handles the movement and rotation of a movable camera in the scene. Movable camera is additional camera that is used to make different views of the scene for recording videos or taking screenshots.

Inherits MonoBehaviour.

Public Attributes

- float movementSpeed = 10f

  `movementSpeed` is the speed at which the camera moves.
- float fastMovementSpeed = 50f

  `fastMovementSpeed` is the speed at which the camera moves when the shift key is held down.
- float rotationSpeed = 5f

  `rotationSpeed` is the speed at which the camera rotates.
- Transform objectToFollow

  `objectToFollow` is the object that the camera will follow.

Private Member Functions

- void Start ()

  The Start method is called before the first frame update. It sets the cameraTransform variable to the camera's Transform component.
- void Update ()

  The Update method is called once per frame. It handles camera movement and rotation based on user input.

Private Attributes

- Transform cameraTransform

  `cameraTransform` is a reference to the camera's Transform component.

4.27.1 Detailed Description

The MovableCameraController class handles the movement and rotation of a movable camera in the scene. Movable camera is additional camera that is used to make different views of the scene for recording videos or taking screenshots.

Definition at line 7 of file MovableCameraController.cs.

4.27.2 Member Function Documentation

4.27.2.1 Start()

```
void MovableCameraController.Start ()  [private]
```

The Start method is called before the first frame update. It sets the cameraTransform variable to the camera's Transform component.

Definition at line 28 of file MovableCameraController.cs.

4.27.2.2 Update()

```
void MovableCameraController.Update ()  [private]
```

The Update method is called once per frame. It handles camera movement and rotation based on user input.

Definition at line 37 of file MovableCameraController.cs.

4.27.3 Member Data Documentation

4.27.3.1 cameraTransform

```
Transform MovableCameraController.cameraTransform  [private]
```

`cameraTransform` is a reference to the camera's Transform component.

Definition at line 10 of file MovableCameraController.cs.

4.27.3.2 fastMovementSpeed

```
float MovableCameraController.fastMovementSpeed = 50f
```

`fastMovementSpeed` is the speed at which the camera moves when the shift key is held down.

Definition at line 17 of file MovableCameraController.cs.

4.27.3.3 movementSpeed

```
float MovableCameraController.movementSpeed = 10f
```

`movementSpeed` is the speed at which the camera moves.

Definition at line 14 of file MovableCameraController.cs.

4.27.3.4 objectToFollow

```
Transform MovableCameraController.objectToFollow
```

`objectToFollow` is the object that the camera will follow.

Definition at line 23 of file MovableCameraController.cs.

4.27.3.5 rotationSpeed

```
float MovableCameraController.rotationSpeed = 5f
```

`rotationSpeed` is the speed at which the camera rotates.

Definition at line 20 of file MovableCameraController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/MovableCameraController.cs

4.28 MultipleTags Class Reference

The MultipleTags class allows a GameObject to have multiple tags.

Inherits MonoBehaviour.

Public Member Functions

- bool HasTag (string tag)

  Checks if a specific tag is assigned to the GameObject.
- void AddTag (string tag)

  Adds a tag to the GameObject.
- void RemoveTag (string tag)

  Removes a tag from the GameObject.
- List< string > GetTags ()

  Returns the list of tags assigned to the GameObject.

Private Attributes

- List< string > tags

  `tags` *is a list of tags assigned to the GameObject.*

4.28.1 Detailed Description

The MultipleTags class allows a GameObject to have multiple tags.

Definition at line 7 of file MultipleTags.cs.

4.28.2 Member Function Documentation

4.28.2.1 AddTag()

```
void MultipleTags.AddTag (
            string tag)
```

Adds a tag to the GameObject.

Parameters

tag	The tag to add.

Definition at line 24 of file MultipleTags.cs.

4.28.2.2 GetTags()

```
List< string > MultipleTags.GetTags ()
```

Returns the list of tags assigned to the GameObject.

Returns

The list of tags.

4.28.2.3 HasTag()

```
bool MultipleTags.HasTag (
            string tag)
```

Checks if a specific tag is assigned to the GameObject.

Parameters

tag	The tag to check.

Returns

True if the tag is assigned to the GameObject, false otherwise.

4.28.2.4 RemoveTag()

```
void MultipleTags.RemoveTag (
            string tag)
```

Removes a tag from the GameObject.

Parameters

tag	The tag to remove.

Definition at line 36 of file MultipleTags.cs.

4.28.3 Member Data Documentation

4.28.3.1 tags

```
List<string> MultipleTags.tags  [private]
```

`tags` is a list of tags assigned to the GameObject.

Definition at line 11 of file MultipleTags.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/MultipleTags.cs

4.29 NPCController Class Reference

The NPCController class handles the interaction between the player and the NPCs in the game.

Inherits MonoBehaviour.

Public Attributes

- Animator animator

  `animator` is the Animator component of the NPC. It is used to control the NPC's animations.
- bool dialogueActive = false

  `dialogueActive` is a flag indicating whether the dialogue is currently active.
- SoundEffectManager soundEffectManager

Private Member Functions

- void Start ()

  It initializes the NPC's Animator component and dialogue lines at the start of the game.
- void Update ()

  It handles the player's interaction with the NPC and the display of the dialogue.
- void OnTriggerEnter (Collider other)

  It checks if the player has entered the NPC's interaction range and, if so, displays the interaction prompt.
- void OnTriggerExit (Collider other)

  It checks if the player has left the NPC's interaction range and, if so, hides the dialogue and interaction prompt.

Private Attributes

- string npcName

  `npcName` *is the name of the NPC.*
- TextAsset fileWithDialogue

  `fileWithDialogue` *is the file containing the dialogue for the NPC.*
- float lineDisplayTimeSec = 2.0f

  `lineDisplayTimeSec` *is the time in seconds each line of dialogue is displayed.*
- TMP_Text canvasText

  `canvasText` *is the TMP_Text object to display the dialogue on the game's canvas.*
- int currentLine = 0

  `currentLine` *is the index of the current line of dialogue being displayed.*
- List< string > dialogue = new List<string>()

  `dialogue` *is the list of dialogue lines for the NPC.*
- bool playerInRange = false

  `playerInRange` *is a flag indicating whether the player is in range to interact with the NPC.*
- float currentLineDisplayTime = 0.0f

  `currentLineDisplayTime` *is the current display time of the line of dialogue.*

4.29.1 Detailed Description

The NPCController class handles the interaction between the player and the NPCs in the game.

Definition at line 8 of file NPCController.cs.

4.29.2 Member Function Documentation

4.29.2.1 OnTriggerEnter()

```
void NPCController.OnTriggerEnter (
            Collider other)  [private]
```

It checks if the player has entered the NPC's interaction range and, if so, displays the interaction prompt.

Definition at line 111 of file NPCController.cs.

4.29.2.2 OnTriggerExit()

```
void NPCController.OnTriggerExit (
            Collider other)  [private]
```

It checks if the player has left the NPC's interaction range and, if so, hides the dialogue and interaction prompt.

Definition at line 129 of file NPCController.cs.

4.29.2.3 Start()

```
void NPCController.Start ()  [private]
```

It initializes the NPC's Animator component and dialogue lines at the start of the game.

Definition at line 49 of file NPCController.cs.

4.29.2.4 Update()

```
void NPCController.Update ()  [private]
```

It handles the player's interaction with the NPC and the display of the dialogue.

Definition at line 63 of file NPCController.cs.

4.29.3 Member Data Documentation

4.29.3.1 animator

```
Animator NPCController.animator
```

`animator` is the Animator component of the NPC. It is used to control the NPC's animations.

Definition at line 27 of file NPCController.cs.

4.29.3.2 canvasText

```
TMP_Text NPCController.canvasText  [private]
```

`canvasText` is the TMP_Text object to display the dialogue on the game's canvas.

Definition at line 24 of file NPCController.cs.

4.29.3.3 currentLine

```
int NPCController.currentLine = 0  [private]
```

`currentLine` is the index of the current line of dialogue being displayed.

Definition at line 30 of file NPCController.cs.

4.29.3.4 currentLineDisplayTime

```
float NPCController.currentLineDisplayTime = 0.0f  [private]
```

`currentLineDisplayTime` is the current display time of the line of dialogue.

Definition at line 42 of file NPCController.cs.

4.29.3.5 dialogue

```
List<string> NPCController.dialogue = new List<string>()  [private]
```

`dialogue` is the list of dialogue lines for the NPC.

Definition at line 33 of file NPCController.cs.

4.29.3.6 dialogueActive

```
bool NPCController.dialogueActive = false
```

`dialogueActive` is a flag indicating whether the dialogue is currently active.

Definition at line 36 of file NPCController.cs.

4.29.3.7 fileWithDialogue

```
TextAsset NPCController.fileWithDialogue  [private]
```

`fileWithDialogue` is the file containing the dialogue for the NPC.

Definition at line 16 of file NPCController.cs.

4.29.3.8 lineDisplayTimeSec

```
float NPCController.lineDisplayTimeSec = 2.0f  [private]
```

`lineDisplayTimeSec` is the time in seconds each line of dialogue is displayed.

Definition at line 20 of file NPCController.cs.

4.29.3.9 npcName

```
string NPCController.npcName  [private]
```

`npcName` is the name of the NPC.

Definition at line 12 of file NPCController.cs.

4.29.3.10 playerInRange

```
bool NPCController.playerInRange = false  [private]
```

`playerInRange` is a flag indicating whether the player is in range to interact with the NPC.

Definition at line 39 of file NPCController.cs.

4.29.3.11 soundEffectManager

```
SoundEffectManager NPCController.soundEffectManager
```

Definition at line 44 of file NPCController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/NPCController.cs

4.30 PlayerAnimationStateController Class Reference

Controls the animation states of the player character based on various game conditions.

Inherits MonoBehaviour.

Public Attributes

- PlayerMovement playerMovement
- LineMovement lineMovement

  Reference to the PlayerMovement script attached to the player object.
- advancedClimbing advancedClimbing

  Reference to the LineMovement script for handling line movement.
- Ziplining ziplining

  Reference to the advancedClimbing script for advanced climbing behaviors.

Private Member Functions

- void Start ()

  Hash for the "xVelocity" parameter in the animator.
- void Update ()

Private Attributes

- Animator animator
- int isCrouchingHash

  Reference to the Ziplining script for handling ziplining mechanics.
- int velocityHash

  Hash for the "isCrouching" parameter in the animator.
- int isJumpingHash

  Hash for the "Velocity" parameter in the animator.
- int isWalkingOnLineHash

  Hash for the "isJumping" parameter in the animator.
- int isWalkingUnderLineHash

  Hash for the "isWalkingOnLine" parameter in the animator.
- int isWalkingUnderLineDirectionHash

  Hash for the "isWalkingUnderLine" parameter in the animator.
- int isHangingHash

  Hash for the "isWalkingUnderLineDirection" parameter in the animator.
- int isZipLiningHash

  Hash for the "isHanging" parameter in the animator.
- int velocityFlatHash

  Hash for the "isZipLining" parameter in the animator.

4.30.1 Detailed Description

Controls the animation states of the player character based on various game conditions.

Definition at line 6 of file PlayerAnimationStateController.cs.

4.30.2 Member Function Documentation

4.30.2.1 Start()

```
void PlayerAnimationStateController.Start ()  [private]
```

Hash for the "xVelocity" parameter in the animator.

Definition at line 28 of file PlayerAnimationStateController.cs.

4.30.2.2 Update()

```
void PlayerAnimationStateController.Update ()  [private]
```

Definition at line 43 of file PlayerAnimationStateController.cs.

4.30.3 Member Data Documentation

4.30.3.1 advancedClimbing

```
advancedClimbing PlayerAnimationStateController.advancedClimbing
```

Reference to the LineMovement script for handling line movement.

Definition at line 14 of file PlayerAnimationStateController.cs.

4.30.3.2 animator

```
Animator PlayerAnimationStateController.animator  [private]
```

Definition at line 8 of file PlayerAnimationStateController.cs.

4.30.3.3 isCrouchingHash

```
int PlayerAnimationStateController.isCrouchingHash  [private]
```

Reference to the Ziplining script for handling ziplining mechanics.

Definition at line 17 of file PlayerAnimationStateController.cs.

4.30.3.4 isHangingHash

```
int PlayerAnimationStateController.isHangingHash  [private]
```

Hash for the "isWalkingUnderLineDirection" parameter in the animator.

Definition at line 23 of file PlayerAnimationStateController.cs.

4.30.3.5 isJumpingHash

```
int PlayerAnimationStateController.isJumpingHash  [private]
```

Hash for the "Velocity" parameter in the animator.

Definition at line 19 of file PlayerAnimationStateController.cs.

4.30.3.6 isWalkingOnLineHash

```
int PlayerAnimationStateController.isWalkingOnLineHash  [private]
```

Hash for the "isJumping" parameter in the animator.

Definition at line 20 of file PlayerAnimationStateController.cs.

4.30.3.7 isWalkingUnderLineDirectionHash

```
int PlayerAnimationStateController.isWalkingUnderLineDirectionHash  [private]
```

Hash for the "isWalkingUnderLine" parameter in the animator.

Definition at line 22 of file PlayerAnimationStateController.cs.

4.30.3.8 isWalkingUnderLineHash

```
int PlayerAnimationStateController.isWalkingUnderLineHash  [private]
```

Hash for the "isWalkingOnLine" parameter in the animator.

Definition at line 21 of file PlayerAnimationStateController.cs.

4.30.3.9 isZipLiningHash

```
int PlayerAnimationStateController.isZipLiningHash  [private]
```

Hash for the "isHanging" parameter in the animator.

Definition at line 24 of file PlayerAnimationStateController.cs.

4.30.3.10 lineMovement

```
LineMovement PlayerAnimationStateController.lineMovement
```

Reference to the PlayerMovement script attached to the player object.

Definition at line 13 of file PlayerAnimationStateController.cs.

4.30.3.11 playerMovement

```
PlayerMovement PlayerAnimationStateController.playerMovement
```

Definition at line 11 of file PlayerAnimationStateController.cs.

4.30.3.12 velocityFlatHash

```
int PlayerAnimationStateController.velocityFlatHash  [private]
```

Hash for the "isZipLining" parameter in the animator.

Definition at line 25 of file PlayerAnimationStateController.cs.

4.30.3.13 velocityHash

```
int PlayerAnimationStateController.velocityHash  [private]
```

Hash for the "isCrouching" parameter in the animator.

Definition at line 18 of file PlayerAnimationStateController.cs.

4.30.3.14 ziplining

```
Ziplining PlayerAnimationStateController.ziplining
```

Reference to the advancedClimbing script for advanced climbing behaviors.

Definition at line 15 of file PlayerAnimationStateController.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/PlayerAnimationStateController.cs

4.31 PlayerData Class Reference

Serializable class representing player data for saving and loading.

Public Member Functions

- PlayerData (int level, int[ ] points)
  Constructor for initializing PlayerData with specific parameters.

Public Attributes

- int currentLevel
  Current level of the player.
- int[ ] points
  Array representing points or scores accumulated by the player.

4.31.1 Detailed Description

Serializable class representing player data for saving and loading.

Definition at line 5 of file PlayerData.cs.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 PlayerData()

```
PlayerData.PlayerData (
            int level,
            int[] points)
```

Constructor for initializing PlayerData with specific parameters.

Parameters

level	The current level of the player.
points	An array of points or scores accumulated by the player.

Definition at line 18 of file PlayerData.cs.

4.31.3 Member Data Documentation

4.31.3.1 currentLevel

```
int PlayerData.currentLevel
```

Current level of the player.

Definition at line 8 of file PlayerData.cs.

4.31.3.2 points

```
int [] PlayerData.points
```

Array representing points or scores accumulated by the player.

Definition at line 11 of file PlayerData.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/PlayerData.cs

4.32 PlayerInLift Class Reference

Manages the player's animation state when inside a lift, based on his movement.

Inherits MonoBehaviour.

Private Member Functions

- void OnCollisionStay (Collision collision)

  Called when the player continuously collides with the lift. It checks the player's movement and updates the animation state accordingly.
- void OnCollisionExit (Collision collision)

  Called when the player exits the collision with the lift. It resets the player's animation state to indicate they are not in the lift.

Private Attributes

- Animator playerAnimator

  `playerAnimator` *is used to control the player's animations.*
- Rigidbody player

  `player` *refers to the player's Rigidbody component, used to check his movement.*

4.32.1 Detailed Description

Manages the player's animation state when inside a lift, based on his movement.

Definition at line 6 of file PlayerInLift.cs.

4.32.2 Member Function Documentation

4.32.2.1 OnCollisionExit()

```
void PlayerInLift.OnCollisionExit (
            Collision collision)  [private]
```

Called when the player exits the collision with the lift. It resets the player's animation state to indicate they are not in the lift.

Parameters

collision	The Collision data associated with this collision.

Definition at line 43 of file PlayerInLift.cs.

4.32.2.2 OnCollisionStay()

```
void PlayerInLift.OnCollisionStay (
            Collision collision)  [private]
```

Called when the player continuously collides with the lift. It checks the player's movement and updates the animation state accordingly.

Parameters

collision	The Collision data associated with this collision.

Definition at line 21 of file PlayerInLift.cs.

4.32.3 Member Data Documentation

4.32.3.1 player

```
Rigidbody PlayerInLift.player  [private]
```

`player` refers to the player's Rigidbody component, used to check his movement.

Definition at line 14 of file PlayerInLift.cs.

4.32.3.2 playerAnimator

```
Animator PlayerInLift.playerAnimator  [private]
```

`playerAnimator` is used to control the player's animations.

Definition at line 10 of file PlayerInLift.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/PlayerInLift.cs

4.33 PlayerMovement Class Reference

Manages player movement including walking, sprinting, crouching, and jumping.

Inherits MonoBehaviour.

Public Member Functions

- void enableAirMovement ()
  Enables air movement for the player.
- void disableAirMovement ()
  Disables air movement for the player.

Public Attributes

- Transform orientation
  Reference to the player's orientation.
- LayerMask isGround
  Layer mask to identify ground surfaces.
- CapsuleCollider capsuleCollider
  Reference to the player's capsule collider.
- float walkSpeedMultiplier
  Multiplier for walking speed.
- float sprintSpeedMultiplier
  Multiplier for sprinting speed.
- float crouchSpeedMultiplier
  Multiplier for crouching speed.
- float moveSpeedLimit
  Limit for the player's movement speed.
- float groundDrag
  Drag applied to the player while on the ground.
- float jumpForce
  Force applied for jumping.
- float jumpCooldown
  Cooldown time between jumps.
- float airMovementMultiplier = 0.8f

Multiplier for movement speed while in the air.

- float velocity

  Current velocity of the player (for UI purposes).
- float velocityFlat

  Current flat (horizontal) velocity of the player (for UI purposes).
- float maxSlopeAngle

  Maximum angle of slope the player can walk on.
- bool touchGround

  Flag indicating if the player is touching the ground.
- bool sprinting = false

  Flag indicating if the player is sprinting.
- bool crouching = false

  Flag indicating if the player is crouching.
- bool dead = false

  Flag indicating if the player is dead.

Private Member Functions

- void Start ()

  Initializes the player movement script.
- void Update ()

  Updates the player state and handles input each frame.
- void FixedUpdate ()

  Handles physics updates for player movement.
- void inputControl ()

  Processes player input for movement and actions.
- void speedLimit ()

  Limits the player's speed to the defined maximum.
- void groundMovement ()

  Handles movement while the player is on the ground.
- void airMovement ()

  Handles movement while the player is in the air.
- void jump ()

  Makes the player jump.
- void afterJump ()

  Resets the player's jump state after jumping.
- bool onSlope ()

  Checks if the player is on a walkable slope.
- bool onSteepSlope ()

  Checks if the player is on a steep slope.
- Vector3 getSlopeMoveDirection ()

  Gets the direction to move on a slope.
- Vector3 getSteepSlopeSlideDirection ()

  Gets the direction to slide down a steep slope.

Private Attributes

- bool airMovementActive = true
- RaycastHit slopeHit
- float horizontalI

  Horizontal input value.
- float verticalI

  Vertical input value.
- float playerHeight

  Height of the player.
- bool readyToJump

  Flag indicating if the player is ready to jump.
- Vector3 moveDir

  Direction of player movement.
- Rigidbody playerRigidbody

  Reference to the player's Rigidbody component.
- float moveSpeedMultiplier
- Quaternion initialRotation

4.33.1 Detailed Description

Manages player movement including walking, sprinting, crouching, and jumping.

Definition at line 7 of file PlayerMovement.cs.

4.33.2 Member Function Documentation

4.33.2.1 afterJump()

```
void PlayerMovement.afterJump ()  [private]
```

Resets the player's jump state after jumping.

Definition at line 267 of file PlayerMovement.cs.

4.33.2.2 airMovement()

```
void PlayerMovement.airMovement ()  [private]
```

Handles movement while the player is in the air.

Definition at line 246 of file PlayerMovement.cs.

4.33.2.3 disableAirMovement()

```
void PlayerMovement.disableAirMovement ()
```

Disables air movement for the player.

Definition at line 288 of file PlayerMovement.cs.

4.33.2.4 enableAirMovement()

```
void PlayerMovement.enableAirMovement ()
```

Enables air movement for the player.

Definition at line 280 of file PlayerMovement.cs.

4.33.2.5 FixedUpdate()

```
void PlayerMovement.FixedUpdate ()  [private]
```

Handles physics updates for player movement.

Definition at line 144 of file PlayerMovement.cs.

4.33.2.6 getSlopeMoveDirection()

```
Vector3 PlayerMovement.getSlopeMoveDirection ()  [private]
```

Gets the direction to move on a slope.

Returns

Normalized direction vector for moving on the slope.

Definition at line 325 of file PlayerMovement.cs.

4.33.2.7 getSteepSlopeSlideDirection()

```
Vector3 PlayerMovement.getSteepSlopeSlideDirection ()  [private]
```

Gets the direction to slide down a steep slope.

Returns

Normalized direction vector for sliding down the steep slope.

Definition at line 334 of file PlayerMovement.cs.

4.33.2.8 groundMovement()

```
void PlayerMovement.groundMovement ()  [private]
```

Handles movement while the player is on the ground.

Definition at line 220 of file PlayerMovement.cs.

4.33.2.9 inputControl()

```
void PlayerMovement.inputControl ()  [private]
```

Processes player input for movement and actions.

Definition at line 162 of file PlayerMovement.cs.

4.33.2.10 jump()

```
void PlayerMovement.jump ()  [private]
```

Makes the player jump.

Definition at line 255 of file PlayerMovement.cs.

4.33.2.11 onSlope()

```
bool PlayerMovement.onSlope ()  [private]
```

Checks if the player is on a walkable slope.

Returns

True if the player is on a slope within the maximum slope angle.

Definition at line 297 of file PlayerMovement.cs.

4.33.2.12 onSteepSlope()

```
bool PlayerMovement.onSteepSlope ()  [private]
```

Checks if the player is on a steep slope.

Returns

True if the player is on a slope steeper than the maximum slope angle.

Definition at line 311 of file PlayerMovement.cs.

4.33.2.13 speedLimit()

```
void PlayerMovement.speedLimit ()  [private]
```

Limits the player's speed to the defined maximum.

Definition at line 207 of file PlayerMovement.cs.

4.33.2.14 Start()

```
void PlayerMovement.Start ()  [private]
```

Initializes the player movement script.

Definition at line 100 of file PlayerMovement.cs.

4.33.2.15 Update()

```
void PlayerMovement.Update ()  [private]
```

Updates the player state and handles input each frame.

Definition at line 112 of file PlayerMovement.cs.

4.33.3 Member Data Documentation

4.33.3.1 airMovementActive

```
bool PlayerMovement.airMovementActive = true  [private]
```

Definition at line 50 of file PlayerMovement.cs.

4.33.3.2 airMovementMultiplier

```
float PlayerMovement.airMovementMultiplier = 0.8f
```

Multiplier for movement speed while in the air.

Definition at line 42 of file PlayerMovement.cs.

4.33.3.3 capsuleCollider

```
CapsuleCollider PlayerMovement.capsuleCollider
```

Reference to the player's capsule collider.

Definition at line 17 of file PlayerMovement.cs.

4.33.3.4 crouching

```
bool PlayerMovement.crouching = false
```

Flag indicating if the player is crouching.

Definition at line 89 of file PlayerMovement.cs.

4.33.3.5 crouchSpeedMultiplier

```
float PlayerMovement.crouchSpeedMultiplier
```

Multiplier for crouching speed.

Definition at line 27 of file PlayerMovement.cs.

4.33.3.6 dead

```
bool PlayerMovement.dead = false
```

Flag indicating if the player is dead.

Definition at line 93 of file PlayerMovement.cs.

4.33.3.7 groundDrag

```
float PlayerMovement.groundDrag
```

Drag applied to the player while on the ground.

Definition at line 33 of file PlayerMovement.cs.

4.33.3.8 horizontalI

```
float PlayerMovement.horizontalI  [private]
```

Horizontal input value.

Definition at line 60 of file PlayerMovement.cs.

4.33.3.9 initialRotation

```
Quaternion PlayerMovement.initialRotation  [private]
```

Definition at line 95 of file PlayerMovement.cs.

4.33.3.10 isGround

```
LayerMask PlayerMovement.isGround
```

Layer mask to identify ground surfaces.

Definition at line 14 of file PlayerMovement.cs.

4.33.3.11 jumpCooldown

```
float PlayerMovement.jumpCooldown
```

Cooldown time between jumps.

Definition at line 39 of file PlayerMovement.cs.

4.33.3.12 jumpForce

```
float PlayerMovement.jumpForce
```

Force applied for jumping.

Definition at line 36 of file PlayerMovement.cs.

4.33.3.13 maxSlopeAngle

```
float PlayerMovement.maxSlopeAngle
```

Maximum angle of slope the player can walk on.

Definition at line 54 of file PlayerMovement.cs.

4.33.3.14 moveDir

```
Vector3 PlayerMovement.moveDir  [private]
```

Direction of player movement.

Definition at line 79 of file PlayerMovement.cs.

4.33.3.15 moveSpeedLimit

```
float PlayerMovement.moveSpeedLimit
```

Limit for the player's movement speed.

Definition at line 30 of file PlayerMovement.cs.

4.33.3.16 moveSpeedMultiplier

```
float PlayerMovement.moveSpeedMultiplier  [private]
```

Definition at line 84 of file PlayerMovement.cs.

4.33.3.17 orientation

```
Transform PlayerMovement.orientation
```

Reference to the player's orientation.

Definition at line 11 of file PlayerMovement.cs.

4.33.3.18 playerHeight

```
float PlayerMovement.playerHeight  [private]
```

Height of the player.

Definition at line 68 of file PlayerMovement.cs.

4.33.3.19 playerRigidbody

```
Rigidbody PlayerMovement.playerRigidbody  [private]
```

Reference to the player's Rigidbody component.

Definition at line 82 of file PlayerMovement.cs.

4.33.3.20 readyToJump

```
bool PlayerMovement.readyToJump  [private]
```

Flag indicating if the player is ready to jump.

Definition at line 76 of file PlayerMovement.cs.

4.33.3.21 slopeHit

```
RaycastHit PlayerMovement.slopeHit  [private]
```

Definition at line 55 of file PlayerMovement.cs.

4.33.3.22 sprinting

```
bool PlayerMovement.sprinting = false
```

Flag indicating if the player is sprinting.

Definition at line 86 of file PlayerMovement.cs.

4.33.3.23 sprintSpeedMultiplier

```
float PlayerMovement.sprintSpeedMultiplier
```

Multiplier for sprinting speed.

Definition at line 24 of file PlayerMovement.cs.

4.33.3.24 touchGround

```
bool PlayerMovement.touchGround
```

Flag indicating if the player is touching the ground.

Definition at line 72 of file PlayerMovement.cs.

4.33.3.25 velocity

```
float PlayerMovement.velocity
```

Current velocity of the player (for UI purposes).

Definition at line 45 of file PlayerMovement.cs.

4.33.3.26 velocityFlat

```
float PlayerMovement.velocityFlat
```

Current flat (horizontal) velocity of the player (for UI purposes).

Definition at line 48 of file PlayerMovement.cs.

4.33.3.27 verticalI

```
float PlayerMovement.verticalI  [private]
```

Vertical input value.

Definition at line 64 of file PlayerMovement.cs.

4.33.3.28 walkSpeedMultiplier

```
float PlayerMovement.walkSpeedMultiplier
```

Multiplier for walking speed.

Definition at line 21 of file PlayerMovement.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/PlayerMovement.cs

4.34 Portal Class Reference

Represents a portal that loads a new scene when triggered by a collider.

Inherits MonoBehaviour.

Public Attributes

- string sceneName

  The name of the scene to load when triggered.

Private Member Functions

- void OnTriggerEnter (Collider other)

  Called when another collider enters the trigger collider attached to this object. Loads the scene specified by sceneName.

4.34.1 Detailed Description

Represents a portal that loads a new scene when triggered by a collider.

Definition at line 7 of file Portal.cs.

4.34.2 Member Function Documentation

4.34.2.1 OnTriggerEnter()

```
void Portal.OnTriggerEnter (
            Collider other)  [private]
```

Called when another collider enters the trigger collider attached to this object. Loads the scene specified by sceneName.

Parameters

other	The Collider other that entered the trigger.

Definition at line 17 of file Portal.cs.

4.34.3 Member Data Documentation

4.34.3.1 sceneName

```
string Portal.sceneName
```

The name of the scene to load when triggered.

Definition at line 10 of file Portal.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Portal.cs

4.35 ResetPlayerAtStart Class Reference

The ResetPlayerAtStart class is responsible for resetting the player's state at the start of the game. It is needed for the player to be correctly moving when on moving platforms.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  Activates and deactivates the player object at the start of the game.

4.35.1 Detailed Description

The ResetPlayerAtStart class is responsible for resetting the player's state at the start of the game. It is needed for the player to be correctly moving when on moving platforms.

Definition at line 7 of file ResetPlayerAtStart.cs.

4.35.2 Member Function Documentation

4.35.2.1 Start()

```
void ResetPlayerAtStart.Start ()  [private]
```

Activates and deactivates the player object at the start of the game.

Definition at line 12 of file ResetPlayerAtStart.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ResetPlayerAtStart.cs

4.36 SaveSystem Class Reference

A static class for handling saving and loading player data using binary serialization.

Static Public Member Functions

- static void Reset ()

  Resets the save data to initial values.
- static void Save (int level, int[ ] points)

  Saves the player data for a specific level.
- static PlayerData Load ()

  Loads the saved player data.
- static void initialize ()

  Initializes the save data if none exists.
- static void updateLevel (int level, int points)

  Updates the level and points in the save data.

4.36.1 Detailed Description

A static class for handling saving and loading player data using binary serialization.

Definition at line 8 of file SaveSystem.cs.

4.36.2 Member Function Documentation

4.36.2.1 initialize()

```
static void SaveSystem.initialize ()  [static]
```

Initializes the save data if none exists.

Definition at line 59 of file SaveSystem.cs.

4.36.2.2 Load()

```
static PlayerData SaveSystem.Load ()  [static]
```

Loads the saved player data.

Returns

The loaded PlayerData object, or null if no save file exists.

Definition at line 38 of file SaveSystem.cs.

4.36.2.3 Reset()

```
static void SaveSystem.Reset ()  [static]
```

Resets the save data to initial values.

Definition at line 13 of file SaveSystem.cs.

4.36.2.4 Save()

```
static void SaveSystem.Save (
            int level,
            int[] points)  [static]
```

Saves the player data for a specific level.

Parameters

level	The level to save data for.
points	An array of points corresponding to different levels.

Definition at line 23 of file SaveSystem.cs.

4.36.2.5 updateLevel()

```
static void SaveSystem.updateLevel (
            int level,
            int points)  [static]
```

Updates the level and points in the save data.

Parameters

level	The level to update.
points	The points to update for the specified level.

Definition at line 80 of file SaveSystem.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/SaveSystem.cs

4.37 ScreenHints Class Reference

The ScreenHints class handles the display of on-screen messages.

Inherits MonoBehaviour.

Public Member Functions

- void LoadMessage (string[ ] messages, string name)

  The LoadMessage method loads a message to be displayed.

Public Attributes

- double timeToDisplay = 3.0

  `timeToDisplay` *is the time in seconds that the message is displayed.*

Private Member Functions

- void Update ()

  Handling the display and removal of messages.

Private Attributes

- string[ ] messages

  `messages` *is an array of strings that form the display message.*
- double timeDisplaying = 0.0

  `timeDisplaying` *is the time in seconds that the current message has been displayed.*
- bool messageShown = false

  `messageShown` *indicates whether the message has been shown.*
- bool isDisplaying = false

  `isDisplaying` *indicates whether any message is currently being displayed.*
- HashSet< string > loadedMessages = new HashSet<string>()

  `loadedMessages` *is a set of the names of the messages that have been loaded.*
- TMP_Text canvasText

  `canvasText` *is the TextMeshPro component where the messages are displayed.*

4.37.1 Detailed Description

The ScreenHints class handles the display of on-screen messages.

Definition at line 9 of file ScreenHints.cs.

4.37.2 Member Function Documentation

4.37.2.1 LoadMessage()

```
void ScreenHints.LoadMessage (
            string[] messages,
            string name)
```

The LoadMessage method loads a message to be displayed.

Parameters

messages	The message to display.
name	The name of the message.

Definition at line 69 of file ScreenHints.cs.

4.37.2.2 Update()

```
void ScreenHints.Update ()  [private]
```

Handling the display and removal of messages.

Definition at line 36 of file ScreenHints.cs.

4.37.3 Member Data Documentation

4.37.3.1 canvasText

```
TMP_Text ScreenHints.canvasText  [private]
```

`canvasText` is the TextMeshPro component where the messages are displayed.

Definition at line 31 of file ScreenHints.cs.

4.37.3.2 isDisplaying

```
bool ScreenHints.isDisplaying = false  [private]
```

`isDisplaying` indicates whether any message is currently being displayed.

Definition at line 24 of file ScreenHints.cs.

4.37.3.3 loadedMessages

```
HashSet<string> ScreenHints.loadedMessages = new HashSet<string>()  [private]
```

`loadedMessages` is a set of the names of the messages that have been loaded.

Definition at line 27 of file ScreenHints.cs.

4.37.3.4 messages

```
string [] ScreenHints.messages  [private]
```

`messages` is an array of strings that form the display message.

Definition at line 12 of file ScreenHints.cs.

4.37.3.5 messageShown

```
bool ScreenHints.messageShown = false  [private]
```

`messageShown` indicates whether the message has been shown.

Definition at line 21 of file ScreenHints.cs.

4.37.3.6 timeDisplaying

```
double ScreenHints.timeDisplaying = 0.0  [private]
```

`timeDisplaying` is the time in seconds that the current message has been displayed.

Definition at line 18 of file ScreenHints.cs.

4.37.3.7 timeToDisplay

```
double ScreenHints.timeToDisplay = 3.0
```

`timeToDisplay` is the time in seconds that the message is displayed.

Definition at line 15 of file ScreenHints.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ScreenHints.cs

4.38 ShowStats Class Reference

Displays level completion information when a player enters a trigger collider.

Inherits MonoBehaviour.

Public Attributes

- TMP_Text levelInfo

  The TextMeshPro Text component used to display level information.
- int level
- int maxPoints

Private Member Functions

- void OnTriggerEnter (Collider other)

  Called when another collider enters the trigger collider attached to this object. Displays level completion status if the entering collider is tagged as "Player".
- void OnTriggerExit (Collider other)

  Called when another collider exits the trigger collider attached to this object. Hides the level information display if the exiting collider is tagged as "Player".

4.38.1 Detailed Description

Displays level completion information when a player enters a trigger collider.

Definition at line 7 of file ShowStats.cs.

4.38.2 Member Function Documentation

4.38.2.1 OnTriggerEnter()

```
void ShowStats.OnTriggerEnter (
            Collider other)  [private]
```

Called when another collider enters the trigger collider attached to this object. Displays level completion status if the entering collider is tagged as "Player".

Parameters

other	The Collider that entered the trigger.

Definition at line 29 of file ShowStats.cs.

4.38.2.2 OnTriggerExit()

```
void ShowStats.OnTriggerExit (
            Collider other)  [private]
```

Called when another collider exits the trigger collider attached to this object. Hides the level information display if the exiting collider is tagged as "Player".

Parameters

other	The Collider that exited the trigger.

Definition at line 52 of file ShowStats.cs.

4.38.3 Member Data Documentation

4.38.3.1 level

```
int ShowStats.level
```

The level number this trigger represents.

Definition at line 17 of file ShowStats.cs.

4.38.3.2 levelInfo

```
TMP_Text ShowStats.levelInfo
```

The TextMeshPro Text component used to display level information.

Definition at line 12 of file ShowStats.cs.

4.38.3.3 maxPoints

```
int ShowStats.maxPoints
```

The maximum points achievable for this level.

Definition at line 22 of file ShowStats.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/ShowStats.cs

4.39 SoundEffectManager Class Reference

Manages sound effects based on various game events and states.

Inherits MonoBehaviour.

Public Attributes

- Animator animator
- checkPointsMenager checkpointsmenager
- PlayerMovement playerMovement
- NPCController npcController
- GoToLastCheckpointOnMine goToLastCheckpointOnMine
- AudioSource source
- AudioClip walking
- AudioClip runing
- AudioClip jumping
- AudioClip hanging
- AudioClip death
- AudioClip talk
- AudioClip explosion
- float walkRunLimit = 10

Private Member Functions

- void Start ()
- void Update ()

Private Attributes

- int lastState
- float oldVelocity
- bool letTalk = false

4.39.1 Detailed Description

Manages sound effects based on various game events and states.

Definition at line 6 of file SoundEffectManager.cs.

4.39.2 Member Function Documentation

4.39.2.1 Start()

```
void SoundEffectManager.Start () [private]
```

Definition at line 94 of file SoundEffectManager.cs.

4.39.2.2 Update()

```
void SoundEffectManager.Update () [private]
```

Definition at line 102 of file SoundEffectManager.cs.

4.39.3 Member Data Documentation

4.39.3.1 animator

```
Animator SoundEffectManager.animator
```

The Animator component controlling character animations.

Definition at line 11 of file SoundEffectManager.cs.

4.39.3.2 checkpointsmenager

```
checkPointsMenager SoundEffectManager.checkpointsmenager
```

Reference to the checkpoint manager script.

Definition at line 16 of file SoundEffectManager.cs.

4.39.3.3 death

```
AudioClip SoundEffectManager.death
```

Sound effect for death.

Definition at line 76 of file SoundEffectManager.cs.

4.39.3.4 explosion

```
AudioClip SoundEffectManager.explosion
```

Sound effect for explosion.

Definition at line 86 of file SoundEffectManager.cs.

4.39.3.5 goToLastCheckpointOnMine

```
GoToLastCheckpointOnMine SoundEffectManager.goToLastCheckpointOnMine
```

Reference to the script handling going to the last checkpoint on mine.

Definition at line 31 of file SoundEffectManager.cs.

4.39.3.6 hanging

```
AudioClip SoundEffectManager.hanging
```

Sound effect for hanging.

Definition at line 71 of file SoundEffectManager.cs.

4.39.3.7 jumping

```
AudioClip SoundEffectManager.jumping
```

Sound effect for jumping.

Definition at line 66 of file SoundEffectManager.cs.

4.39.3.8 lastState

```
int SoundEffectManager.lastState  [private]
```

Stores the last animator state hash.

Definition at line 36 of file SoundEffectManager.cs.

4.39.3.9 letTalk

```
bool SoundEffectManager.letTalk = false  [private]
```

Flag indicating if the NPC is currently talking.

Definition at line 46 of file SoundEffectManager.cs.

4.39.3.10 npcController

```
NPCController SoundEffectManager.npcController
```

Reference to the NPC controller script.

Definition at line 26 of file SoundEffectManager.cs.

4.39.3.11 oldVelocity

```
float SoundEffectManager.oldVelocity  [private]
```

Stores the previous velocity of the player.

Definition at line 41 of file SoundEffectManager.cs.

4.39.3.12 playerMovement

```
PlayerMovement SoundEffectManager.playerMovement
```

Reference to the player movement script.

Definition at line 21 of file SoundEffectManager.cs.

4.39.3.13 runing

```
AudioClip SoundEffectManager.runing
```

Sound effect for running.

Definition at line 61 of file SoundEffectManager.cs.

4.39.3.14 source

```
AudioSource SoundEffectManager.source
```

The AudioSource component used to play sound effects.

Definition at line 51 of file SoundEffectManager.cs.

4.39.3.15 talk

```
AudioClip SoundEffectManager.talk
```

Sound effect for NPC dialogue.

Definition at line 81 of file SoundEffectManager.cs.

4.39.3.16 walking

```
AudioClip SoundEffectManager.walking
```

Sound effect for walking.

Definition at line 56 of file SoundEffectManager.cs.

4.39.3.17 walkRunLimit

```
float SoundEffectManager.walkRunLimit = 10
```

The velocity threshold distinguishing between walking and running.

Definition at line 91 of file SoundEffectManager.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/SoundEffectManager.cs

4.40 StickyPlatform Class Reference

The StickyPlatform class handles the interaction between the player and sticky platforms in the game. When the player is on a sticky platform, he become a child of the platform, moving with it. When the player stops colliding with the platform, he are no longer a child of the platform.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  It initializes the objectToStick with the player's GameObject.
- void OnCollisionStay (Collision collision)

  Checks if the objectToStick is colliding with the platform. If the colliding object is the objectToStick, it becomes a child of the platform.
- void OnCollisionExit (Collision collision)

  Checks if the objectToStick stopped colliding with the platform. If the object that stopped colliding is the objectToStick, it is no longer a child of the platform.

Private Attributes

- GameObject objectToStick

  `objectToStick` *is the GameObject that will stick to the platform.*

4.40.1 Detailed Description

The StickyPlatform class handles the interaction between the player and sticky platforms in the game. When the player is on a sticky platform, he become a child of the platform, moving with it. When the player stops colliding with the platform, he are no longer a child of the platform.

Definition at line 8 of file StickyPlatform.cs.

4.40.2 Member Function Documentation

4.40.2.1 OnCollisionExit()

```
void StickyPlatform.OnCollisionExit (
            Collision collision)  [private]
```

Checks if the objectToStick stopped colliding with the platform. If the object that stopped colliding is the objectTo↩Stick, it is no longer a child of the platform.

Definition at line 37 of file StickyPlatform.cs.

4.40.2.2 OnCollisionStay()

```
void StickyPlatform.OnCollisionStay (
            Collision collision)  [private]
```

Checks if the objectToStick is colliding with the platform. If the colliding object is the objectToStick, it becomes a child of the platform.

Definition at line 25 of file StickyPlatform.cs.

4.40.2.3 Start()

```
void StickyPlatform.Start ()  [private]
```

It initializes the objectToStick with the player's GameObject.

Definition at line 16 of file StickyPlatform.cs.

4.40.3 Member Data Documentation

4.40.3.1 objectToStick

```
GameObject StickyPlatform.objectToStick  [private]
```

`objectToStick` is the GameObject that will stick to the platform.

Definition at line 11 of file StickyPlatform.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/StickyPlatform.cs

4.41 VelocityText Class Reference

Displays the current player velocity using TextMeshProUGUI.

Inherits MonoBehaviour.

Public Attributes

- GameObject textMeshProVelocity
- float playerVelocity

Private Member Functions

- void Start ()
- void Update ()

Private Attributes

- TextMeshProUGUI textMeshProVelocityText

  The TextMeshProUGUI component used to display the player velocity.
- PlayerMovement playerMovement

4.41.1 Detailed Description

Displays the current player velocity using TextMeshProUGUI.

Definition at line 7 of file VelocityText.cs.

4.41.2 Member Function Documentation

4.41.2.1 Start()

```
void VelocityText.Start ()  [private]
```

Definition at line 28 of file VelocityText.cs.

4.41.2.2 Update()

```
void VelocityText.Update ()  [private]
```

Definition at line 34 of file VelocityText.cs.

4.41.3 Member Data Documentation

4.41.3.1 playerMovement

```
PlayerMovement VelocityText.playerMovement  [private]
```

Definition at line 25 of file VelocityText.cs.

4.41.3.2 playerVelocity

```
float VelocityText.playerVelocity
```

The current velocity of the player.

Definition at line 17 of file VelocityText.cs.

4.41.3.3 textMeshProVelocity

```
GameObject VelocityText.textMeshProVelocity
```

The GameObject containing the TextMeshProUGUI component for displaying velocity.

Definition at line 12 of file VelocityText.cs.

4.41.3.4 textMeshProVelocityText

```
TextMeshProUGUI VelocityText.textMeshProVelocityText  [private]
```

The TextMeshProUGUI component used to display the player velocity.

Definition at line 22 of file VelocityText.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/VelocityText.cs

4.42 Wallrunning Class Reference

Enables wall running mechanics for the player character.

Inherits MonoBehaviour.

Public Attributes

- LayerMask whatIsWall
- LayerMask whatIsGround
- float wallRunForce
- float maxWallRunTime
- float wallCheckDistance
- float minJumpHeight
- Transform orientation
- bool wallrunning

Private Member Functions

- void Start ()
- void Update ()
- void FixedUpdate ()
- bool CanWallRun ()

  Checks if the player can start wall running based on jump height and ground detection.
- void WallRunningMovement ()

  Applies forces to the Rigidbody to simulate wall running movement.

Private Attributes

- float horizontalInput
- float verticalInput
- RaycastHit leftWallhit
- RaycastHit rightWallhit
- bool wallLeft
- bool wallRight
- Rigidbody rb

4.42.1 Detailed Description

Enables wall running mechanics for the player character.

Definition at line 6 of file Wallrunning.cs.

4.42.2 Member Function Documentation

4.42.2.1 CanWallRun()

```
bool Wallrunning.CanWallRun ()  [private]
```

Checks if the player can start wall running based on jump height and ground detection.

Returns

True if the player can start wall running, false otherwise.

Definition at line 118 of file Wallrunning.cs.

4.42.2.2 FixedUpdate()

```
void Wallrunning.FixedUpdate ()  [private]
```

Definition at line 109 of file Wallrunning.cs.

4.42.2.3 Start()

```
void Wallrunning.Start ()  [private]
```

Definition at line 86 of file Wallrunning.cs.

4.42.2.4 Update()

```
void Wallrunning.Update ()  [private]
```

Definition at line 91 of file Wallrunning.cs.

4.42.2.5 WallRunningMovement()

```
void Wallrunning.WallRunningMovement ()  [private]
```

Applies forces to the Rigidbody to simulate wall running movement.

Definition at line 126 of file Wallrunning.cs.

4.42.3 Member Data Documentation

4.42.3.1 horizontalInput

```
float Wallrunning.horizontalInput  [private]
```

Horizontal input axis value.

Definition at line 32 of file Wallrunning.cs.

4.42.3.2 leftWallhit

```
RaycastHit Wallrunning.leftWallhit  [private]
```

RaycastHit structure for the left wall detection.

Definition at line 53 of file Wallrunning.cs.

4.42.3.3 maxWallRunTime

```
float Wallrunning.maxWallRunTime
```

Maximum duration allowed for wall running.

Definition at line 27 of file Wallrunning.cs.

4.42.3.4 minJumpHeight

```
float Wallrunning.minJumpHeight
```

Minimum height for a jump to be considered valid for wall running.

Definition at line 48 of file Wallrunning.cs.

4.42.3.5 orientation

```
Transform Wallrunning.orientation
```

Reference to the orientation transform.

Definition at line 74 of file Wallrunning.cs.

4.42.3.6 rb

```
Rigidbody Wallrunning.rb  [private]
```

Reference to the Rigidbody component attached to the player.

Definition at line 84 of file Wallrunning.cs.

4.42.3.7 rightWallhit

```
RaycastHit Wallrunning.rightWallhit  [private]
```

RaycastHit structure for the right wall detection.

Definition at line 58 of file Wallrunning.cs.

4.42.3.8 verticalInput

```
float Wallrunning.verticalInput  [private]
```

Vertical input axis value.

Definition at line 37 of file Wallrunning.cs.

4.42.3.9 wallCheckDistance

```
float Wallrunning.wallCheckDistance
```

Distance to check for walls.

Definition at line 43 of file Wallrunning.cs.

4.42.3.10 wallLeft

```
bool Wallrunning.wallLeft  [private]
```

Boolean indicating if there is a wall detected on the left.

Definition at line 63 of file Wallrunning.cs.

4.42.3.11 wallRight

```
bool Wallrunning.wallRight  [private]
```

Boolean indicating if there is a wall detected on the right.

Definition at line 68 of file Wallrunning.cs.

4.42.3.12 wallRunForce

```
float Wallrunning.wallRunForce
```

Force applied when wall running.

Definition at line 22 of file Wallrunning.cs.

4.42.3.13 wallrunning

```
bool Wallrunning.wallrunning
```

Indicates if the player is currently wall running.

Definition at line 79 of file Wallrunning.cs.

4.42.3.14 whatIsGround

```
LayerMask Wallrunning.whatIsGround
```

Layer mask to determine what is considered as ground for detecting jump height.

Definition at line 17 of file Wallrunning.cs.

4.42.3.15 whatIsWall

```
LayerMask Wallrunning.whatIsWall
```

Layer mask to determine what is considered as a wall for wall running.

Definition at line 12 of file Wallrunning.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Wallrunning.cs

4.43 WaypointsFollower Class Reference

The WaypointsFollower class handles the movement of an object along a set of waypoints.

Inherits MonoBehaviour.

Private Member Functions

- void Start ()

  The Start method is called before after game starts. It checks if the object has a Rigidbody component and if so, multiplies the movingSpeed by 4.
- void Update ()

  The Update method called once per frame. It moves the object towards the next waypoint and updates the next↩ Waypoint index when the object reaches a waypoint.
- void OnDrawGizmosSelected ()

  Method to draw lines between waypoints in the Scene view to help visualize the path the object will take.

Private Attributes

- float movingSpeed

  movingSpeed is the speed at which the object moves between waypoints.
- Transform[ ] waypoints

  waypoints is an array of Transform objects that the object will move between.
- int nextWaypoint = 0

  nextWaypoint is the index of the next waypoint in the waypoints array that the object will move towards.
- bool rotateTowardsWaypoint = false

  rotateTowardsWaypoint is a boolean that determines whether the object should rotate to face the next waypoint.

4.43.1 Detailed Description

The WaypointsFollower class handles the movement of an object along a set of waypoints.

Definition at line 6 of file WaypointsFollower.cs.

4.43.2 Member Function Documentation

4.43.2.1 OnDrawGizmosSelected()

```
void WaypointsFollower.OnDrawGizmosSelected ()  [private]
```

Method to draw lines between waypoints in the Scene view to help visualize the path the object will take.

Definition at line 59 of file WaypointsFollower.cs.

4.43.2.2 Start()

```
void WaypointsFollower.Start ()  [private]
```

The Start method is called before after game starts. It checks if the object has a Rigidbody component and if so, multiplies the movingSpeed by 4.

Definition at line 26 of file WaypointsFollower.cs.

4.43.2.3 Update()

```
void WaypointsFollower.Update ()  [private]
```

The Update method called once per frame. It moves the object towards the next waypoint and updates the next↩ Waypoint index when the object reaches a waypoint.

Definition at line 37 of file WaypointsFollower.cs.

4.43.3 Member Data Documentation

4.43.3.1 movingSpeed

```
float WaypointsFollower.movingSpeed  [private]
```

`movingSpeed` is the speed at which the object moves between waypoints.

Definition at line 10 of file WaypointsFollower.cs.

4.43.3.2 nextWaypoint

```
int WaypointsFollower.nextWaypoint = 0  [private]
```

`nextWaypoint` is the index of the next waypoint in the waypoints array that the object will move towards.

Definition at line 17 of file WaypointsFollower.cs.

4.43.3.3 rotateTowardsWaypoint

```
bool WaypointsFollower.rotateTowardsWaypoint = false  [private]
```

`rotateTowardsWaypoint` is a boolean that determines whether the object should rotate to face the next waypoint.

Definition at line 21 of file WaypointsFollower.cs.

4.43.3.4 waypoints

```
Transform [] WaypointsFollower.waypoints  [private]
```

`waypoints` is an array of Transform objects that the object will move between.

Definition at line 14 of file WaypointsFollower.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/WaypointsFollower.cs

4.44 Ziplining Class Reference

The Ziplining class handles the player's interaction with ziplines in the game.

Inherits MonoBehaviour.

Public Member Functions

- bool IsZiplining ()

  Checks if the player is currently ziplining.

Public Attributes

- Camera3P camera3P

  `camera3P` *is reference to the Camera3P component, which is used to block the player's rotation when moving on the line.*
- bool isZiplining = false

  `isZiplining` *is a flag indicating whether the player is currently ziplining.*

Private Member Functions

- void Start ()

  It initializes the player's Rigidbody, ScreenHints, and PlayerMovement components at the start of the game.
- void calculateLineDirection (Collision collision)

  Calculates the direction of the zipline based on the collision.
- void OnCollisionEnter (Collision collision)

  Checks if the player is in contact with a zipline. It sets the ziplining.
- void OnCollisionExit (Collision collision)

  Handles the player's disengagement from the zipline.
- void Update ()

  Handles the player's movement along the zipline and disengagement from the zipline.

Private Attributes

- float speed = 3.0f

  `speed` *is the speed at which the player moves along the zipline.*
- Rigidbody playerRigidbody

  `playerRigidbody` *is the player's Rigidbody component.*
- Vector3 startPoint

  `startPoint` *is the start point needed to calculate the direction of the zipline.*
- Vector3 endPoint

  `endPoint` *is the end point needed to calculate the direction of the zipline.*
- Vector3 endLine

  `endLine` *is the end (lowest) point of the zipline.*
- PlayerMovement playerMovement

  `playerMovement` *is the player's movement component.*
- Vector3 lineDirection

  `lineDirection` *is the direction of the line.*
- readonly string[ ] ziplineButtonsPrompts

  `ziplineButtonsPrompts` *is the button prompt displayed to the player when they are ziplining.*
- ScreenHints buttonPromptsController

  `buttonPromptsController` *is the controller for the button prompts displayed to the player.*

4.44.1 Detailed Description

The Ziplining class handles the player's interaction with ziplines in the game.

Definition at line 7 of file Ziplining.cs.

4.44.2 Member Function Documentation

4.44.2.1 calculateLineDirection()

```
void Ziplining.calculateLineDirection (
            Collision collision)  [private]
```

Calculates the direction of the zipline based on the collision.

Definition at line 62 of file Ziplining.cs.

4.44.2.2 IsZiplining()

```
bool Ziplining.IsZiplining ()
```

Checks if the player is currently ziplining.

Definition at line 170 of file Ziplining.cs.

4.44.2.3 OnCollisionEnter()

```
void Ziplining.OnCollisionEnter (
            Collision collision)  [private]
```

Checks if the player is in contact with a zipline. It sets the ziplining.

Definition at line 102 of file Ziplining.cs.

4.44.2.4 OnCollisionExit()

```
void Ziplining.OnCollisionExit (
            Collision collision)  [private]
```

Handles the player's disengagement from the zipline.

Definition at line 130 of file Ziplining.cs.

4.44.2.5 Start()

```
void Ziplining.Start ()  [private]
```

It initializes the player's Rigidbody, ScreenHints, and PlayerMovement components at the start of the game.

Definition at line 52 of file Ziplining.cs.

4.44.2.6 Update()

```
void Ziplining.Update ()  [private]
```

Handles the player's movement along the zipline and disengagement from the zipline.

Definition at line 146 of file Ziplining.cs.

4.44.3 Member Data Documentation

4.44.3.1 buttonPromptsController

```
ScreenHints Ziplining.buttonPromptsController  [private]
```

`buttonPromptsController` is the controller for the button prompts displayed to the player.

Definition at line 47 of file Ziplining.cs.

4.44.3.2 camera3P

```
Camera3P Ziplining.camera3P
```

`camera3P` is reference to the Camera3P component, which is used to block the player's rotation when moving on the line.

Definition at line 17 of file Ziplining.cs.

4.44.3.3 endLine

```
Vector3 Ziplining.endLine  [private]
```

`endLine` is the end (lowest) point of the zipline.

Definition at line 32 of file Ziplining.cs.

4.44.3.4 endPoint

```
Vector3 Ziplining.endPoint  [private]
```

`endPoint` is the end point needed to calculate the direction of the zipline.

Definition at line 29 of file Ziplining.cs.

4.44.3.5 isZiplining

```
bool Ziplining.isZiplining = false
```

`isZiplining` is a flag indicating whether the player is currently ziplining.

Definition at line 23 of file Ziplining.cs.

4.44.3.6 lineDirection

```
Vector3 Ziplining.lineDirection  [private]
```

`lineDirection` is the direction of the line.

Definition at line 38 of file Ziplining.cs.

4.44.3.7 playerMovement

```
PlayerMovement Ziplining.playerMovement  [private]
```

`playerMovement` is the player's movement component.

Definition at line 35 of file Ziplining.cs.

4.44.3.8 playerRigidbody

```
Rigidbody Ziplining.playerRigidbody  [private]
```

`playerRigidbody` is the player's Rigidbody component.

Definition at line 20 of file Ziplining.cs.

4.44.3.9 speed

```
float Ziplining.speed = 3.0f  [private]
```

`speed` is the speed at which the player moves along the zipline.

Definition at line 12 of file Ziplining.cs.

4.44.3.10 startPoint

```
Vector3 Ziplining.startPoint  [private]
```

`startPoint` is the start point needed to calculate the direction of the zipline.

Definition at line 26 of file Ziplining.cs.

4.44.3.11 zipliningButtonsPrompts

```
readonly string [] Ziplining.zipliningButtonsPrompts  [private]
```

Initial value:

```
=
    {
        "press <sprite name=\"E\"> to let go"
    }
```

`ziplineButtonsPrompts` is the button prompt displayed to the player when they are ziplining.

Definition at line 41 of file Ziplining.cs.

The documentation for this class was generated from the following file:

- Assets/Scripts/Ziplining.cs

Chapter 5

File Documentation

5.1 Assets/Scripts/advancedClimbing.cs File Reference

Classes

- class advancedClimbing

  This class handles advanced climbing mechanics, including line movement and ziplining.

5.2 advancedClimbing.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class advancedClimbing : MonoBehaviour
00007 {
00008     [Header("references")]
00010     Rigidbody rigidbody;
00012     public Camera cam;
00013
00014     [Header("variables")]
00016     public float afterHandleJumpForce;
00018     public float sphereCastR;
00020     public float hitingLenght;
00022     public float handleAfterJumpDelay;
00024     public LayerMask Handler;
00026     RaycastHit hit;
00027
00029     private LineMovement lineMovement;
00031     private Ziplining ziplining;
00032
00034     public bool canHandle;
00036     public bool stopHolding;
00037
00041     void Start()
00042     {
00043         rigidbody = GetComponent<Rigidbody>();
00044         lineMovement = GetComponent<LineMovement>();
00045         ziplining = GetComponent<Ziplining>();
00046
00047         canHandle = false;
00048         stopHolding = false;
00049     }
00050
00054     void Update()
00055     {
00056         if (canHandle)
00057         {
00058             if (rigidbody.useGravity == true)
00059             {
00060                 rigidbody.useGravity = false;
00061                 rigidbody.velocity = Vector3.zero;
```

```
00062             }
00063
00064             if (Input.GetKeyDown(KeyCode.Space))
00065             {
00066                 canHandle = false;
00067                 rigidbody.useGravity = true;
00068                 rigidbody.AddForce(cam.gameObject.transform.forward * afterHandleJumpForce,
      ForceMode.Impulse);
00069             }
00070         }
00071         else if (!lineMovement.IsMovingOnLine() && !ziplining.IsZiplining())
00072         {
00073             if (rigidbody.useGravity == false)
00074             {
00075                 canHandle = false;
00076                 rigidbody.useGravity = true;
00077             }
00078         }
00079     }
00080
00085     private void OnCollisionEnter(Collision other)
00086     {
00087         if (other.gameObject.layer == 8)
00088         {
00089             // Get the position of the object that was collided with
00090             Vector3 collisionPosition = other.transform.position;
00091
00092             // Get the position of the object to which this script is attached
00093             Vector3 objectPosition = transform.position;
00094
00095             // Calculate the direction to the object that was collided with
00096             Vector3 dir = collisionPosition - objectPosition;
00097
00098             // Set the y value to 0 to focus only on rotation around the y-axis
00099             dir.y = 0;
00100
00101             // Calculate the new rotation
00102             Quaternion rot = Quaternion.LookRotation(dir);
00103
00104             // Set the new rotation
00105             transform.rotation = rot;
00106
00107             canHandle = true;
00108         }
00109     }
00110
00115     private void OnCollisionExit(Collision other)
00116     {
00117         if (other.gameObject.layer == 8)
00118         {
00119             canHandle = false;
00120         }
00121     }
00122 }
```

5.3 Assets/Scripts/AvalibleIfLevel.cs File Reference

Classes

- class AvalibleIfLevel

  This class controls the activation of a GameObject based on the player's current level.

5.4 AvalibleIfLevel.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class AvalibleIfLevel : MonoBehaviour
00007 {
00009     public int levelToActive;
00010
00014     void Start()
00015     {
```

```
00016         if (SaveSystem.Load().currentLevel < levelToActive)
00017         {
00018             gameObject.SetActive(false);
00019         }
00020     }
00021 }
```

5.5 Assets/Scripts/Camera3P.cs File Reference

Classes

- class Camera3P

  This class controls the third-person camera behavior, including aiming and default camera modes.

5.6 Camera3P.cs

Go to the documentation of this file.

```
00001 using Cinemachine;
00002 using UnityEngine;
00003 using UnityEngine.UI;
00004
00008 public class Camera3P : MonoBehaviour
00009 {
00010     [Header("ObjectsRef")]
00012     public Transform orientation;
00014     public Transform player;
00016     public Transform playerObject;
00018     public Camera cam;
00019
00020     [Header("Variables")]
00022     public float rotationSpeed;
00024     public Transform lookDir;
00025
00027     [SerializeField]
00028     float horizontalI;
00030     [SerializeField]
00031     float verticalI;
00033     public bool aim;
00035     public Image aimImage;
00036
00038     public bool disableRotation;
00039
00041     public CinemachineFreeLook defaultCinemachine;
00043     public CinemachineFreeLook aimCinemachine;
00044
00048     void Start()
00049     {
00050         aim = false;
00051         aimImage.enabled = aim;
00052         defaultCinemachine.Priority = 1;
00053         aimCinemachine.Priority = 0;
00054     }
00055
00059     void Update()
00060     {
00061         if (Input.GetKeyDown(KeyCode.F))
00062         {
00063             aim = !aim;
00064
00065             if (aim)
00066             {
00067                 defaultCinemachine.Priority = 0;
00068                 aimCinemachine.Priority = 1;
00069             }
00070             else
00071             {
00072                 defaultCinemachine.Priority = 1;
00073                 aimCinemachine.Priority = 0;
00074             }
00075             aimImage.enabled = aim;
00076         }
00077
00078         if (!aim)
```

```
00079         {
00080             Vector3 viewDir = player.position - new Vector3(transform.position.x, player.position.y,
      transform.position.z);
00081             orientation.forward = viewDir.normalized;
00082             horizontalI = Input.GetAxis("Horizontal");
00083             verticalI = Input.GetAxis("Vertical");
00084 
00085             Vector3 inputDir = (orientation.forward * verticalI) + (orientation.right * horizontalI);
00086             if (inputDir.magnitude != 0 && !disableRotation)
00087             {
00088                 playerObject.forward = Vector3.Slerp(playerObject.forward, inputDir.normalized,
      Time.deltaTime * rotationSpeed);
00089             }
00090         }
00091         else
00092         {
00093             Vector3 lookAtDir = (lookDir.position - new Vector3(transform.position.x,
      lookDir.position.y, transform.position.z)).normalized;
00094             orientation.forward = lookAtDir;
00095             playerObject.forward = lookAtDir;
00096         }
00097     }
00098 }
```

5.7 Assets/Scripts/CheckPoint.cs File Reference

Classes

- class CheckPoint

  This class handles checkpoint functionality, saving the player's position when they reach a checkpoint.

5.8 CheckPoint.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 
00006 public class CheckPoint : MonoBehaviour
00007 {
00009     public checkPointsMenager menager;
00010 
00015     private void OnTriggerEnter(Collider other)
00016     {
00017         Debug.Log("Check point saved");
00018         menager.setPosition(new Vector3(transform.position.x, transform.position.y,
      transform.position.z));
00019         Destroy(gameObject);
00020     }
00021 }
```

5.9 Assets/Scripts/checkPointsMenager.cs File Reference

Classes

- class checkPointsMenager

  This class manages checkpoints and handles the player's position after death.

5.10 checkPointsMenager.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class checkPointsMenager : MonoBehaviour
00007 {
00009     Vector3 AfterDeadPosition;
00011     public Transform Player;
00013     public bool trigger = false;
00014
00018     void Start()
00019     {
00020         AfterDeadPosition = Player.position;
00021     }
00022
00027     public Vector3 getPosition()
00028     {
00029         trigger = true;
00030         return AfterDeadPosition;
00031     }
00032
00037     public void setPosition(Vector3 position)
00038     {
00039         AfterDeadPosition = position;
00040     }
00041
00045     private void Update()
00046     {
00047
00048     }
00049 }
```

5.11 Assets/Scripts/Climbing.cs File Reference

Classes

- class Climbing

  Handles the climbing mechanics for the player.

5.12 Climbing.cs

Go to the documentation of this file.

```
00001 using System;
00002 using UnityEngine;
00003
00007 public class Climbing : MonoBehaviour
00008 {
00009     [Header("References")]
00011     [SerializeField]
00012     Transform orientation;
00014     [SerializeField]
00015     LayerMask climbableWall;
00016
00017     [SerializeField]
00019     Camera3P camera3P;
00020
00021     [Header("Climbing")]
00023     [SerializeField]
00024     float climbSpeed = 1.5f;
00026     [SerializeField]
00027     float maxClimbAngle = 30.0f;
00028
00030     private float originalDrag;
00032     private bool isClimbing = false;
00033
00034     [Header("Climbing Timer")]
00036     [SerializeField]
00037     float maxClimbTimeX = 0.4f;
00039     [SerializeField]
00040     float maxClimbTimeZ = 0.4f;
```

```
00041
00043     private RaycastHit hitWall;
00044
00046     private readonly string[] climbingButtonsPrompts =
00047     {
00048         "press <sprite name=\"E\"> to let go",
00049         "move with <sprite name=\"W\"> <sprite name=\"A\"> <sprite name=\"S\"> <sprite name=\"D\">"
00050     };
00051
00053     ScreenHints buttonPromptsController;
00054
00056     private float climbTimeX = 0.0f;
00057     private float climbTimeZ = 0.0f;
00058
00060     PlayerMovement playerMovement;
00061
00063     private bool climbingLock = false;
00064
00066     Rigidbody playerRigidbody;
00067
00069     Animator animator;
00070
00074     void Start()
00075     {
00076         playerRigidbody = GetComponent<Rigidbody>();
00077         buttonPromptsController = GetComponent<ScreenHints>();
00078         playerMovement = GetComponent<PlayerMovement>();
00079         animator = transform.Find("Ch24_nonPBR").GetComponent<Animator>();
00080     }
00081
00085     void Update()
00086     {
00087         SetClimbingState();
00088         if (isClimbing)
00089         {
00090             originalDrag = playerRigidbody.drag;
00091             Climb();
00092
00093             camera3P.disableRotation = true;
00094             playerMovement.disableAirMovement();
00095         }
00096     }
00097
00101     private void DisableClimbing()
00102     {
00103         playerRigidbody.useGravity = true;
00104
00105         if (!playerMovement.touchGround)
00106         {
00107             playerRigidbody.drag = 0;
00108         }
00109         else
00110         {
00111             playerRigidbody.drag = originalDrag;
00112         }
00113
00114         animator.SetBool("isClimbingUp", false);
00115         animator.SetBool("isClimbingDown", false);
00116         animator.SetBool("isFallingFromWall", true);
00117         camera3P.disableRotation = false;
00118         playerMovement.enableAirMovement();
00119     }
00120
00124     private void Climb()
00125     {
00126         playerRigidbody.useGravity = false;
00127         playerRigidbody.drag = 0;
00128
00129         // Get the normal of the wall hit
00130         Vector3 wallNormal = hitWall.normal;
00131
00132         // Set the rotation of Ch24_nonPBR to face opposite the wall instantly
00133         transform.Find("Ch24_nonPBR").rotation = Quaternion.LookRotation(-wallNormal, Vector3.up);
00134
00135
00136         // Calculate movement direction perpendicular to the wall
00137         Vector3 rightDirection = Vector3.Cross(Vector3.up, wallNormal).normalized;
00138         Vector3 forwardDirection = Vector3.Cross(wallNormal, rightDirection).normalized;
00139
00140         // Calculate player's movement based on input
00141         Vector3 moveDirection = Vector3.zero;
00142
00143         // Get input and set movement direction
00144         if (Input.GetKey(KeyCode.W))
00145         {
00146             moveDirection += forwardDirection;
00147             animator.SetBool("isClimbingUp", true);
```

```
00148             animator.SetBool("isClimbingDown", false);
00149             animator.SetBool("isClimbingLeft", false);
00150             animator.SetBool("isClimbingRight", false);
00151         }
00152 
00153         if (Input.GetKey(KeyCode.S))
00154         {
00155             moveDirection -= forwardDirection;
00156             animator.SetBool("isClimbingUp", false);
00157             animator.SetBool("isClimbingDown", true);
00158             animator.SetBool("isClimbingLeft", false);
00159             animator.SetBool("isClimbingRight", false);
00160         }
00161 
00162         if (Input.GetKey(KeyCode.A))
00163         {
00164             moveDirection += rightDirection;
00165             animator.SetBool("isClimbingUp", false);
00166             animator.SetBool("isClimbingDown", false);
00167             animator.SetBool("isClimbingLeft", true);
00168             animator.SetBool("isClimbingRight", false);
00169         }
00170 
00171         if (Input.GetKey(KeyCode.D))
00172         {
00173             moveDirection -= rightDirection;
00174             animator.SetBool("isClimbingUp", false);
00175             animator.SetBool("isClimbingDown", false);
00176             animator.SetBool("isClimbingLeft", false);
00177             animator.SetBool("isClimbingRight", true);
00178         }
00179         // Apply movement along the wall
00180         playerRigidbody.velocity = moveDirection.normalized * climbSpeed;
00181 
00182         if (moveDirection == Vector3.zero)
00183         {
00184             animator.SetBool("isFallingFromWall", false);
00185             animator.SetBool("isClimbingUp", false);
00186             animator.SetBool("isClimbingDown", false);
00187             animator.SetBool("isClimbingLeft", false);
00188             animator.SetBool("isClimbingRight", false);
00189         }
00190 
00191         // Release climbing
00192         if (Input.GetKey(KeyCode.E))
00193         {
00194             DisableClimbing();
00195             isClimbing = false;
00196             climbingLock = true;
00197         }
00198 
00199         climbTimeX += Time.deltaTime;
00200         climbTimeZ += Time.deltaTime;
00201 
00202         if (climbTimeX > maxClimbTimeX)
00203         {
00204             playerRigidbody.velocity = new Vector3(0.0f, playerRigidbody.velocity.y,
      playerRigidbody.velocity.z);
00205             climbTimeX = 0.0f;
00206         }
00207         if (climbTimeZ > maxClimbTimeZ)
00208         {
00209             playerRigidbody.velocity = new Vector3(playerRigidbody.velocity.x,
      playerRigidbody.velocity.y, 0.0f);
00210             climbTimeZ = 0.0f;
00211         }
00212     }
00213 
00214 
00218     private void SetClimbingState()
00219     {
00220         bool onWall = Physics.SphereCast(transform.position, 0.3f, orientation.forward, out hitWall,
      1f, climbableWall);
00221         float angle = Vector3.Angle(orientation.forward, -hitWall.normal);
00222 
00223         SetClimbingLock();
00224 
00225         if (!onWall && isClimbing)
00226         {
00227             DisableClimbing();
00228             isClimbing = false;
00229         }
00230         else if (!playerMovement.touchGround && onWall && angle <= maxClimbAngle && !climbingLock)
00231         {
00232             isClimbing = true;
00233             buttonPromptsController.LoadMessage(climbingButtonsPrompts, "climbing");
00234         }
```

```
00235         else if (isClimbing)
00236         {
00237             DisableClimbing();
00238             isClimbing = false;
00239         }
00240         else
00241         {
00242             isClimbing = false;
00243             //camera3P.disableRotation = false;
00244         }
00245     }
00246
00250     private void SetClimbingLock()
00251     {
00252         if (climbingLock)
00253         {
00254             if (playerMovement.touchGround)
00255             {
00256                 climbingLock = false;
00257             }
00258             else if (Input.GetKey(KeyCode.W))
00259             {
00260                 climbingLock = false;
00261             }
00262         }
00263     }
00264 }
```

5.13 Assets/Scripts/Collectible.cs File Reference

Classes

- class Collectible

  This class represents counting the number of collectibles the player has collected.

5.14 Collectible.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class Collectible : MonoBehaviour
00007 {
00009     public int collectedCounter = 0;
00010
00012     LevelStatistics levelStatistics;
00013
00017     void Start()
00018     {
00019         levelStatistics = GetComponent<LevelStatistics>();
00020     }
00021
00028     private void OnTriggerEnter(Collider other)
00029     {
00030         if (other.CompareTag("Collectible"))
00031         {
00032             Destroy(other.gameObject);
00033             collectedCounter++;
00034             levelStatistics.collectedCount = collectedCounter;
00035         }
00036     }
00037 }
```

5.15 Assets/Scripts/ColliderFromMesh.cs File Reference

Classes

- class ColliderFromMesh

  This class creates a MeshCollider from a SkinnedMeshRenderer and updates it each frame.

5.16 ColliderFromMesh.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class ColliderFromMesh : MonoBehaviour
00007 {
00009     private MeshCollider meshCollider;
00011     private SkinnedMeshRenderer skinnedMeshRenderer;
00013     private Mesh colliderMesh;
00014
00018     void Start()
00019     {
00020         // Get the SkinnedMeshRenderer from the object
00021         skinnedMeshRenderer = GetComponent<SkinnedMeshRenderer>();
00022         if (skinnedMeshRenderer == null)
00023         {
00024             Debug.LogError("SkinnedMeshRenderer not found on the object.");
00025             return;
00026         }
00027
00028         // Create a new mesh to hold the current skin position
00029         colliderMesh = new Mesh();
00030
00031         // Add MeshCollider to the object
00032         meshCollider = gameObject.AddComponent<MeshCollider>();
00033     }
00034
00038     void Update()
00039     {
00040         // Bake the current mesh into colliderMesh
00041         skinnedMeshRenderer.BakeMesh(colliderMesh);
00042
00043         // Set the baked mesh as the sharedMesh in MeshCollider
00044         meshCollider.sharedMesh = colliderMesh;
00045
00046         // Ensure the collider has the correct position and rotation
00047         meshCollider.transform.position = skinnedMeshRenderer.transform.position;
00048         meshCollider.transform.rotation = skinnedMeshRenderer.transform.rotation;
00049     }
00050 }
```

5.17 Assets/Scripts/Dash.cs File Reference

Classes

- class Dash

  This class handles the dash ability for the player, including cooldown management and movement limits.

5.18 Dash.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class Dash : MonoBehaviour
00007 {
00008     [Header("References")]
00010     private PlayerMovement movement;
00012     public Transform cam;
00014     private Rigidbody rb;
00016     public GameObject playerObject;
00017
00018     [Header("Variables")]
00020     public float fullCooldown = 1.5f;
00022     public float activeCooldown = 0f;
00024     public float dashForce = 800f;
00026     public float dashLimit = 80f;
00028     public float standardLimit = 25f;
00029
00033     void Start()
00034     {
```

```
00035         rb = GetComponent<Rigidbody>();
00036         movement = GetComponent<PlayerMovement>();
00037     }
00038
00042     void Update()
00043     {
00044         if (activeCooldown > 0)
00045         {
00046             activeCooldown -= Time.deltaTime;
00047         }
00048         else
00049         {
00050             if (Input.GetKeyDown(KeyCode.C))
00051             {
00052                 DashAbility();
00053             }
00054         }
00055     }
00056
00060     private void DashAbility()
00061     {
00062         movement.moveSpeedLimit = dashLimit;
00063         Vector3 cameraEulerAngles = cam.eulerAngles;
00064         Vector3 newRotation = new Vector3(transform.eulerAngles.x, cameraEulerAngles.y,
      transform.eulerAngles.z);
00065         playerObject.transform.eulerAngles = newRotation;
00066
00067         Vector3 dashVector = cam.forward * dashForce;
00068         rb.velocity = Vector3.zero;
00069         rb.AddForce(new Vector3(dashVector.x, 0, dashVector.z), ForceMode.Impulse);
00070         activeCooldown = fullCooldown;
00071         Invoke(nameof(resetLimit), fullCooldown);
00072     }
00073
00077     private void resetLimit()
00078     {
00079         movement.moveSpeedLimit = standardLimit;
00080     }
00081 }
```

5.19 Assets/Scripts/EnemyMovement.cs File Reference

Classes

- class EnemyMovement

  This class handles enemy movement and behavior, including interaction with the player and responding to being kicked.

5.20 EnemyMovement.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using UnityEngine.AI;
00003
00007 public class EnemyMovement : MonoBehaviour
00008 {
00010     public NavMeshAgent agent;
00012     private bool playerInRange;
00014     private bool fighting;
00016     public Transform playerTransform;
00018     private bool stand;
00019
00021     private bool isKicked;
00022
00024     public Animator animator;
00025
00029     private void Update()
00030     {
00031         if (isKicked)
00032         {
00033             agent.SetDestination(transform.position);
00034             return;
00035         }
```

```
00036
00037         if (playerInRange)
00038         {
00039             animator.SetBool("isInRange", true);
00040         }
00041         else
00042         {
00043             animator.SetBool("isInRange", false);
00044         }
00045
00046         if (fighting)
00047         {
00048             animator.SetBool("fight", true);
00049         }
00050         else
00051         {
00052             animator.SetBool("fight", false);
00053         }
00054     }
00055
00060     private void OnTriggerStay(Collider other)
00061     {
00062         if (isKicked)
00063             return;
00064
00065         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00066         {
00067             float dist = Vector3.Distance(other.transform.position, transform.position);
00068
00069             if (dist > 1)
00070             {
00071                 fighting = false;
00072                 playerInRange = true;
00073                 Debug.Log("Enemy set agro");
00074                 AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
00075                 int stateHash = stateInfo.shortNameHash;
00076                 if (stateHash == Animator.StringToHash("Standing Melee Attack Horizontal"))
00077                 {
00078                     agent.SetDestination(transform.position);
00079                 }
00080                 else
00081                 {
00082                     agent.SetDestination(playerTransform.position);
00083                 }
00084             }
00085             else
00086             {
00087                 agent.SetDestination(transform.position);
00088                 fighting = true;
00089             }
00090         }
00091     }
00092
00097     private void OnTriggerExit(Collider other)
00098     {
00099         playerInRange = false;
00100     }
00101
00105     public void GetKicked()
00106     {
00107         Debug.Log("Enemy got kicked");
00108         animator.SetBool("isKicked", true);
00109         isKicked = true;
00110         agent.SetDestination(transform.position); // Stop moving
00111     }
00112 }
```

5.21 Assets/Scripts/FinishLevelBarrel.cs File Reference

Classes

- class FinishLevelBarrel

  This class handles the finish level logic when a barrel enters the trigger.

5.22 FinishLevelBarrel.cs

Go to the documentation of this file.

```
00001 using System.Collections;
00002 using System.Collections.Generic;
00003 using UnityEngine;
00004
00008 public class FinishLevelBarrel : MonoBehaviour
00009 {
00014     private void OnTriggerEnter(Collider collider)
00015     {
00016         if (collider.CompareTag("Barrel"))
00017         {
00018             gameObject.layer = 9;
00019         }
00020     }
00021 }
```

5.23 Assets/Scripts/FPSTarget.cs File Reference

Classes

- class FPSTarget

  This class is used to set the target frame rate for the application.

5.24 FPSTarget.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class FPSTarget : MonoBehaviour
00007 {
00009     private int targetFrameRate = 144;
00010
00015     private void Start()
00016     {
00017         QualitySettings.vSyncCount = 0;
00018         Application.targetFrameRate = targetFrameRate;
00019     }
00020 }
```

5.25 Assets/Scripts/GoToLastCheckpoint.cs File Reference

Classes

- class GoToLastCheckpoint

  This class handles resetting the player to the last checkpoint upon collision or trigger with specific objects.

5.26 GoToLastCheckpoint.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class GoToLastCheckpoint : MonoBehaviour
00007 {
00009     public checkPointsMenager menager;
00011     public GameObject player;
00013     public Animator animator;
00014
00019     private void OnTriggerEnter(Collider other)
00020     {
00021         Debug.Log("reset");
00022
00023         if (other.CompareTag("Player"))
00024         {
00025             player.transform.position = menager.getPosition();
00026             Debug.Log("colission");
00027         }
00028     }
00029
00034     private void OnCollisionEnter(Collision collision)
00035     {
00036         Debug.Log("reset");
00037
00038         if (collision.gameObject.CompareTag("Player"))
00039         {
00040             if (!animator)
00041             {
00042                 player.transform.position = menager.getPosition();
00043                 Debug.Log("colission");
00044             }
00045             else
00046             {
00047                 AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
00048                 int stateHash = stateInfo.shortNameHash;
00049                 if (stateHash == Animator.StringToHash("Standing Melee Attack Horizontal"))
00050                 {
00051                     player.transform.position = menager.getPosition();
00052                     Debug.Log("colission");
00053                 }
00054             }
00055         }
00056     }
00057 }
```

5.27 Assets/Scripts/GoToLastCheckpointOnMine.cs File Reference

Classes

- class GoToLastCheckpointOnMine

  This class manages respawning the player and a barrel at the last checkpoint upon collision with specific objects.

5.28 GoToLastCheckpointOnMine.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using System.Collections;
00003
00007 public class GoToLastCheckpointOnMine : MonoBehaviour
00008 {
00010     public checkPointsMenager menager;
00012     public GameObject player;
00014     public GameObject barrel;
00016     public PlayerMovement playerMovement;
00018     public Animator animator;
00020     public GameObject explosion;
00022     public bool exploding = false;
00024     private int isExplodingHash;
00025
```

```
00029     void Start()
00030     {
00031         isExplodingHash = Animator.StringToHash("isExploding");
00032     }
00033
00038     private void OnTriggerEnter(Collider other)
00039     {
00040         Debug.Log("reset");
00041
00042         if (other.CompareTag("Player"))
00043         {
00044             StartCoroutine(RespawnPlayerWithDelay());
00045         }
00046         if (other.CompareTag("Barrel"))
00047         {
00048             StartCoroutine(RespawnBarrelWithDelay());
00049         }
00050     }
00051
00056     private void OnCollisionEnter(Collision collision)
00057     {
00058         Debug.Log("reset");
00059
00060         if (collision.gameObject.CompareTag("Player"))
00061         {
00062             StartCoroutine(RespawnPlayerWithDelay());
00063         }
00064         if (collision.gameObject.CompareTag("Barrel"))
00065         {
00066             StartCoroutine(RespawnBarrelWithDelay());
00067         }
00068     }
00069
00073     private IEnumerator RespawnPlayerWithDelay()
00074     {
00075         exploding = true;
00076         playerMovement.enabled = false;
00077         player.transform.rotation = Quaternion.identity;
00078         explosion.transform.position = player.transform.position;
00079         explosion.transform.rotation = player.transform.rotation;
00080         explosion.SetActive(true);
00081         playerMovement.dead = true;
00082         animator.SetBool(isExplodingHash, true);
00083         yield return new WaitForSeconds(3f);
00084
00085         playerMovement.velocity = 0f;
00086         player.transform.position = menager.getPosition();
00087         Debug.Log("colission");
00088         explosion.SetActive(false);
00089         animator.SetBool(isExplodingHash, false);
00090         playerMovement.dead = false;
00091         playerMovement.enabled = true;
00092     }
00093
00097     private IEnumerator RespawnBarrelWithDelay()
00098     {
00099         exploding = true;
00100         explosion.transform.position = barrel.transform.position;
00101         explosion.transform.rotation = barrel.transform.rotation;
00102         explosion.SetActive(true);
00103         barrel.SetActive(false);
00104         yield return new WaitForSeconds(3f);
00105         explosion.SetActive(false);
00106         barrel.SetActive(true);
00107         barrel.gameObject.transform.position = new Vector3(56.5f, 6, 122);
00108         barrel.gameObject.transform.rotation = Quaternion.Euler(new Vector3(0, 0, 90));
00109     }
00110 }
```

5.29 Assets/Scripts/Grappling.cs File Reference

Classes

- class Grappling

  The Grappling class handles the grappling mechanics in the game.

5.30 Grappling.cs

Go to the documentation of this file.

```
00001 using Unity.VisualScripting;
00002 using UnityEngine;
00003
00007 public class Grappling : MonoBehaviour
00008 {
00010     private LineRenderer lineRenderer;
00011
00013     private Vector3 grapplePoint;
00014
00016     private Transform gunTip;
00017
00019     private bool grappling = false;
00020
00022     [SerializeField]
00023     private float maxGrapplingDistance;
00024
00026     [SerializeField]
00027     private Transform playerCamera;
00028
00030     [SerializeField]
00031     private Transform playerObject;
00032
00034     [SerializeField]
00035     private float spring = 10.0f;
00036
00038     [SerializeField]
00039     private float damper = 7f;
00040
00042     [SerializeField]
00043     private float massScale = 4.5f;
00044
00046     private MultipleTags tags;
00047
00049     private string checkTag = "grappable";
00050
00054     void Start()
00055     {
00056         lineRenderer = GetComponent<LineRenderer>();
00057         gunTip = transform.Find("GunTip");
00058         lineRenderer.positionCount = 0;
00059     }
00060
00064     void Update()
00065     {
00066         if (Input.GetMouseButtonDown(0))
00067         {
00068             StartGrappling();
00069         }
00070         else if (Input.GetMouseButtonUp(0))
00071         {
00072             StopGrappling();
00073         }
00074     }
00075
00079     private void LateUpdate()
00080     {
00081         if (grappling)
00082         {
00083             DrawLine();
00084         }
00085     }
00086
00090     private void DrawLine()
00091     {
00092         lineRenderer.SetPosition(0, gunTip.position);
00093         lineRenderer.SetPosition(1, grapplePoint);
00094     }
00095
00099     private void StartGrappling()
00100     {
00101         if (Physics.Raycast(playerCamera.position, playerCamera.forward, out RaycastHit hitPoint,
      maxGrapplingDistance, ~(1 << LayerMask.NameToLayer("Player"))))
00102         {
00103             hitPoint.transform.TryGetComponent<MultipleTags>(out var multipleTags);
00104             if (multipleTags != null)
00105                 if (multipleTags.HasTag(checkTag))
00106                 {
00107                     grapplePoint = hitPoint.point;
00108                     grappling = true;
00109                     lineRenderer.positionCount = 2;
00110
00111                     SpringJoint springJoint = playerObject.AddComponent<SpringJoint>();
```

```
00112                     springJoint.autoConfigureConnectedAnchor = false;
00113                     springJoint.connectedAnchor = grapplePoint;
00114                     springJoint.maxDistance = Vector3.Distance(playerObject.position, grapplePoint) *
      0.8f;
00115                     springJoint.minDistance = Vector3.Distance(playerObject.position, grapplePoint) *
      0.1f;
00116                     springJoint.spring = spring;
00117                     springJoint.damper = damper;
00118                     springJoint.massScale = massScale;
00119                 }
00120             }
00121     }
00122
00126     private void StopGrappling()
00127     {
00128         grappling = false;
00129         lineRenderer.positionCount = 0;
00130         Destroy(playerObject.GetComponent<SpringJoint>());
00131     }
00132 }
```

5.31 Assets/Scripts/InteractionWithObjects.cs File Reference

Classes

- class InteractionWithObjects

  This class manages interactions with objects based on raycasting from the player's position.

5.32 InteractionWithObjects.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using UnityEngine.SceneManagement;
00003
00007 public class InteractionWithObjects : MonoBehaviour
00008 {
00010     [Header("References")]
00011     public GameObject Interactinfo;
00013     public LayerMask interactable;
00015     public LayerMask defaultLayer;
00017     public GameObject playerObject;
00019     public string interaction = "animate";
00021     public LevelStatistics ls;
00023     public int levelUnlocked;
00024
00026     [Header("Variables")]
00027     public float rayDistance;
00028
00029     private RaycastHit hit;
00030
00031     // Update is called once per frame
00032     void Update()
00033     {
00034         if (Physics.Raycast(playerObject.transform.position, playerObject.transform.forward, out hit,
      rayDistance, interactable))
00035         {
00036             Debug.Log("can interact");
00037             Interactinfo.gameObject.SetActive(true);
00038             GameObject hitted = hit.collider.transform.root.gameObject;
00039             Animator animator = hitted.GetComponent<Animator>();
00040             if (Input.GetKeyDown(KeyCode.E))
00041             {
00042                 if (interaction == "animate")
00043                 {
00044                     animator.SetTrigger("interact");
00045                     hitted.layer = defaultLayer;
00046                 }
00047                 else if (interaction == "exit map")
00048                 {
00049                     SaveSystem.updateLevel(levelUnlocked, ls.collectedCount);
00050                     SceneManager.LoadScene("MainLocation");
00051                 }
00052             }
```

```
00053         }
00054         else
00055         {
00056             Interactinfo.gameObject.SetActive(false);
00057         }
00058     }
00059 }
```

5.33 Assets/Scripts/kickEnemy.cs File Reference

Classes

- class kickEnemy

  This class manages kicking enemies when they collide with a trigger.

5.34 kickEnemy.cs

Go to the documentation of this file.

```
00001 using System.Collections;
00002 using System.Collections.Generic;
00003 using UnityEngine;
00004
00008 public class kickEnemy : MonoBehaviour
00009 {
00011     public Animator playerAnimator;
00013     private bool getKicked;
00014
00015     // Start is called before the first frame update
00016     void Start()
00017     {
00018         getKicked = false;
00019     }
00020
00025     void OnTriggerEnter(Collider other)
00026     {
00027         if (other.CompareTag("Enemy"))
00028         {
00029             AnimatorStateInfo stateInfo = playerAnimator.GetCurrentAnimatorStateInfo(0);
00030             int stateHash = stateInfo.shortNameHash;
00031             Debug.Log("kick enemy");
00032
00033             if (stateHash == Animator.StringToHash("Mma Kick"))
00034             {
00035                 getKicked = true;
00036
00037                 EnemyMovement enemyMovement = other.GetComponent<EnemyMovement>();
00038                 if (enemyMovement != null)
00039                 {
00040                     enemyMovement.GetKicked();
00041                 }
00042             }
00043         }
00044     }
00045 }
```

5.35 Assets/Scripts/Level4WaterReset.cs File Reference

Classes

- class Level4WaterReset

  The Level4WaterReset class is responsible for resetting the player's position when the player is underwater. It checks if the player is underwater and if so, starts a countdown. If the player stays underwater for a certain amount of time, the player's position is reset.

5.36 Level4WaterReset.cs

Go to the documentation of this file.

```
00001 using TMPro;
00002 using UnityEngine;
00003
00009 public class Level4WaterReset : MonoBehaviour
00010 {
00012     private double waterLevel;
00013
00015     private double playerHeight;
00016
00018     [SerializeField]
00019     private TMP_Text underWaterText;
00020
00022     private GameObject player;
00023
00025     [SerializeField]
00026     private Transform startPoint;
00027
00029     [SerializeField]
00030     private float numberOfSecondsToWait = 5;
00031
00033     private float timeElapsed = 0;
00034
00036     private int secondsElapsed = 0;
00037
00041     void Start()
00042     {
00043         waterLevel = GameObject.Find("Water Specular").transform.position.y;
00044         player = GameObject.Find("Player");
00045         playerHeight = player.transform.Find("Ch24_nonPBR").GetComponent<CapsuleCollider>().height;
00046     }
00047
00051     void Update()
00052     {
00053         if (waterLevel > (player.transform.position.y + playerHeight / 2) + 0.1)
00054         {
00055             timeElapsed += Time.deltaTime;
00056             secondsElapsed = (int)timeElapsed % 60;
00057             underWaterText.text = "Underwater!\n" + (numberOfSecondsToWait - secondsElapsed) +
    "\nseconds to reset";
00058             underWaterText.gameObject.SetActive(true);
00059             if (timeElapsed >= numberOfSecondsToWait)
00060             {
00061                 ResetPlayerPosition();
00062                 timeElapsed = 0;
00063             }
00064         }
00065         else
00066         {
00067             underWaterText.gameObject.SetActive(false);
00068             timeElapsed = 0;
00069         }
00070     }
00071
00075     private void ResetPlayerPosition()
00076     {
00077         var rigidBody = player.GetComponent<Rigidbody>();
00078         rigidBody.velocity = Vector3.zero;
00079         player.transform.position = startPoint.position;
00080         rigidBody.MovePosition(startPoint.position);
00081     }
00082 }
```

5.37 Assets/Scripts/Level5LavaReset.cs File Reference

Classes

- class Level5LavaReset

  The Level5LavaReset class is responsible for resetting the player's position when the player fell into the lava. It checks if the player fell into lava and if so, starts a countdown. After a certain amount of time, the player's position is reset.

5.38 Level5LavaReset.cs

Go to the documentation of this file.

```
00001 using TMPro;
00002
00003 using UnityEngine;
00004
00010 public class Level5LavaReset : MonoBehaviour
00011 {
00013     private double lavaLevel;
00014
00016     private double playerHeight;
00017
00019     [SerializeField]
00020     private TMP_Text textGUI;
00021
00023     private GameObject player;
00024
00026     [SerializeField]
00027     private Transform startPoint;
00028
00030     [SerializeField]
00031     private float numberOfSecondsToWait = 5;
00032
00034     [SerializeField]
00035     private GameObject lavaObject;
00036
00038     private float timeElapsed = 0;
00039
00041     private int secondsElapsed = 0;
00042
00046     void Start()
00047     {
00048         player = GameObject.Find("Player");
00049         playerHeight = player.transform.position.y;
00050         lavaLevel = lavaObject.transform.position.y;
00051     }
00052
00056     void Update()
00057     {
00058         if (player.transform.position.y - playerHeight / 2 - 0.2f <= lavaLevel)
00059         {
00060             player.transform.Find("Ch24_nonPBR/PlayerFire").gameObject.SetActive(true);
00061             timeElapsed += Time.deltaTime;
00062             secondsElapsed = (int)timeElapsed % 60;
00063             textGUI.text = (numberOfSecondsToWait - secondsElapsed) + "\nseconds to reset";
00064             textGUI.gameObject.SetActive(true);
00065             if (timeElapsed >= numberOfSecondsToWait)
00066             {
00067                 ResetPlayerPosition();
00068                 timeElapsed = 0;
00069             }
00070         }
00071         else
00072         {
00073             textGUI.gameObject.SetActive(false);
00074             timeElapsed = 0;
00075         }
00076     }
00077
00081     private void ResetPlayerPosition()
00082     {
00083         player.transform.Find("Ch24_nonPBR/PlayerFire").gameObject.SetActive(false);
00084         var rigidBody = player.GetComponent<Rigidbody>();
00085         rigidBody.velocity = Vector3.zero;
00086         player.transform.position = startPoint.position;
00087         rigidBody.MovePosition(startPoint.position);
00088     }
00089 }
```

5.39 Assets/Scripts/LevelStatistics.cs File Reference

Classes

- class LevelStatistics

  The LevelStatistics class handles the tracking and display of collectible items in a level.

5.40 LevelStatistics.cs

Go to the documentation of this file.

```
00001 using TMPro;
00002 using UnityEngine;
00003
00007 public class LevelStatistics : MonoBehaviour
00008 {
00010     public int collectedCount = 0;
00011
00013     [SerializeField]
00014     public int totalCollectibleCount = 0;
00015
00017     [SerializeField]
00018     private TMP_Text canvasText;
00019
00021     private string[] messages =
00022     {
00023         "Level Statistics",
00024         ""
00025     };
00026
00030     void Start()
00031     {
00032         totalCollectibleCount = GameObject.FindGameObjectsWithTag("Collectible").Length;
00033     }
00034
00038     void LateUpdate()
00039     {
00040         messages[1] = "Collected items: " + collectedCount + " / " + totalCollectibleCount;
00041         canvasText.text = "";
00042         foreach (string message in messages)
00043         {
00044             canvasText.text += message + "\n";
00045         }
00046     }
00047 }
```

5.41 Assets/Scripts/Lift.cs File Reference

Classes

- class Lift

  The Lift class handles the movement of the lift in the game.

5.42 Lift.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class Lift : MonoBehaviour
00007 {
00011     [SerializeField]
00012     private float speed = 0.008f;
00013
00017     private GameObject platformUpPost;
00018
00022     private GameObject platformDownPost;
00023
00027     private Renderer chain1Renderer;
00028
00032     private Renderer chain2Renderer;
00033
00037     private Rigidbody platform;
00038
00042     private bool isPlatformUp = true;
00043
00047     private bool isPlatformMoving = false;
00048
00052     void Start()
00053     {
```

```
00054         platformUpPost = transform.Find("LiftPlatformUpPos").gameObject;
00055         platformDownPost = transform.Find("LiftPlatformDownPos").gameObject;
00056 
00057         platform = transform.Find("LiftPlatform").GetComponent<Rigidbody>();
00058 
00059         chain1Renderer = transform.Find("LiftPillar1/LiftChainPillar1").GetComponent<Renderer>();
00060         chain2Renderer = transform.Find("LiftPillar2/LiftChainPillar2").GetComponent<Renderer>();
00061     }
00062 
00066     void Update()
00067     {
00068         if (isPlatformMoving)
00069         {
00070             if (isPlatformUp)
00071             {
00072                 if (Vector3.Distance(platform.transform.position, platformDownPost.transform.position)
      < 0.1f)
00073                 {
00074                     isPlatformUp = false;
00075                     isPlatformMoving = false;
00076                 }
00077                 else
00078                 {
00079                     platform.MovePosition(Vector3.MoveTowards(platform.transform.position,
      platformDownPost.transform.position, speed));
00080                     UpdateTextureOffset();
00081                 }
00082             }
00083             else
00084             {
00085                 if (Vector3.Distance(platform.transform.position, platformUpPost.transform.position) <
      0.1f)
00086                 {
00087                     isPlatformUp = true;
00088                     isPlatformMoving = false;
00089                 }
00090                 else
00091                 {
00092                     platform.MovePosition(Vector3.MoveTowards(platform.transform.position,
      platformUpPost.transform.position, speed));
00093                     UpdateTextureOffset();
00094                 }
00095             }
00096         }
00097     }
00098 
00102     private void UpdateTextureOffset()
00103     {
00104         float oldYOffset = chain1Renderer.material.mainTextureOffset.y;
00105         if (oldYOffset >= 1000)
00106         {
00107             oldYOffset = 0;
00108         }
00109         Vector2 newOffset = new Vector2(0, oldYOffset + 0.5f);
00110 
00111         chain1Renderer.material.mainTextureOffset = newOffset;
00112         chain2Renderer.material.mainTextureOffset = newOffset;
00113     }
00114 
00118     public void ActivateLift()
00119     {
00120         if (!isPlatformMoving)
00121         {
00122             isPlatformMoving = true;
00123         }
00124     }
00125 
00129     public void MoveLiftUp()
00130     {
00131         if (!isPlatformMoving && !isPlatformUp)
00132         {
00133             isPlatformMoving = true;
00134         }
00135     }
00136 
00140     public void MoveLiftDown()
00141     {
00142         if (!isPlatformMoving && isPlatformUp)
00143         {
00144             isPlatformMoving = true;
00145         }
00146     }
00147 }
```

5.43 Assets/Scripts/LiftActivation.cs File Reference

Classes

- class LiftActivation

  The LiftActivation class handles the activation of lifts in the game when the player enters a specific trigger area.

5.44 LiftActivation.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class LiftActivation : MonoBehaviour
00007 {
00013     private void OnTriggerEnter(Collider other)
00014     {
00015         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00016         {
00017             Lift lift;
00018             if (transform.name == "LiftPlatformActivationPos")
00019             {
00020                 lift = transform.parent.parent.GetComponent<Lift>();
00021             }
00022             else
00023             {
00024                 lift = transform.parent.GetComponent<Lift>();
00025             }
00026
00027             if (lift != null)
00028             {
00029                 if (transform.name == "LiftUpActivationPos")
00030                 {
00031                     lift.MoveLiftUp();
00032                 }
00033                 else if (transform.name == "LiftDownActivationPos")
00034                 {
00035                     lift.MoveLiftDown();
00036                 }
00037                 else
00038                 {
00039                     lift.ActivateLift();
00040                 }
00041             }
00042         }
00043     }
00044 }
```

5.45 Assets/Scripts/LineMovement.cs File Reference

Classes

- class LineMovement

  The LineMovement class handles the player's movement along movable lines in the game.

Typedefs

- using Vector3 = UnityEngine.Vector3

5.45.1 Typedef Documentation

5.45.1.1 Vector3

```
using Vector3 = UnityEngine.Vector3
```

Definition at line 2 of file LineMovement.cs.

5.46 LineMovement.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using Vector3 = UnityEngine.Vector3;
00003
00007 public class LineMovement : MonoBehaviour
00008 {
00009     [Header("Moving Timer")]
00011     private float moveTime = 0.0f;
00013     private float moveTimeMax = 0.1f;
00014
00015     [Header("Speed")]
00016     [SerializeField]
00018     private float speed = 3.0f;
00019
00021     public bool isMovingOnLine = false;
00022
00023     [Header("References")]
00024     [SerializeField]
00026     private Transform playerOrientation;
00027
00028     [SerializeField]
00030     public Camera3P camera3P;
00031
00033     private PlayerMovement playerMovement;
00034
00036     private Vector3 lineDirection;
00037
00039     ScreenHints buttonPromptsController;
00040
00042     private readonly string[] movingAboveLineButtonPrompts =
00043     {
00044         "press <sprite name=\"W\"> and <sprite name=\"S\"> to move forward and backward"
00045     };
00046
00048     private readonly string[] movingUnderLineButtonPrompts =
00049     {
00050         "press <sprite name=\"E\"> to let go",
00051         "press <sprite name=\"W\"> and <sprite name=\"S\"> to move forward and backward"
00052     };
00053
00055     private Rigidbody playerRigidbody;
00056
00058     public int direction = 1;
00059
00061     public bool isAboveLine = false;
00062
00064     private Quaternion initialPlayerRotation;
00065
00067     Transform minimapIcon;
00068
00070     private int minimapRotChangeY = 90;
00071
00075     private void calculateLineDirection(Collision collision)
00076     {
00077         CapsuleCollider lineCollider = collision.gameObject.GetComponent<CapsuleCollider>();
00078         if (lineCollider == null) return; // Early exit if the collider is not found
00079
00080         // Calculate the line's direction based on its orientation
00081         Vector3 lineVector = Vector3.zero;
00082         switch (lineCollider.direction)
00083         {
00084             case 0: // X-axis
00085                 lineVector = collision.transform.right;
00086                 break;
00087             case 1: // Y-axis
00088                 lineVector = collision.transform.up;
00089                 break;
```

```
00090             case 2: // Z-axis
00091                 lineVector = collision.transform.forward;
00092                 break;
00093         }
00094 
00095         // Calculate the start and end points of the line
00096         Vector3 startPoint = collision.transform.position - (lineVector * lineCollider.height / 2);
00097         Vector3 endPoint = collision.transform.position + (lineVector * lineCollider.height / 2);
00098 
00099         // Adjust for the collider's center offset
00100         startPoint += lineCollider.center;
00101         endPoint += lineCollider.center;
00102 
00103         // Calculate the direction of the line
00104         lineDirection = (endPoint - startPoint).normalized;
00105     }
00106 
00111     private void OnCollisionEnter(Collision collision)
00112     {
00113         if (collision.gameObject.CompareTag("movableLine"))
00114         {
00115             calculateLineDirection(collision);
00116             if (collision.contacts.Length > 0)
00117             {
00118                 initialPlayerRotation = transform.rotation;
00119                 playerMovement.disableAirMovement();
00120                 ContactPoint contact = collision.GetContact(0);
00121                 float dotUp = Vector3.Dot(contact.normal, Vector3.up);
00122                 float dotDown = Vector3.Dot(contact.normal, Vector3.down);
00123                 isAboveLine = dotUp > 0.7f;
00124                 bool isUnderLine = dotDown > 0.9f;
00125                 isMovingOnLine = isUnderLine || isAboveLine;
00126 
00127                 if (isMovingOnLine)
00128                 {
00129                     playerRigidbody.useGravity = false;
00130                     playerRigidbody.drag = 0;
00131                 }
00132 
00133                 if (isAboveLine)
00134                 {
00135                     buttonPromptsController.LoadMessage(movingAboveLineButtonPrompts,
      "aboveLineMovement");
00136                 }
00137                 else if (isUnderLine)
00138                 {
00139                     buttonPromptsController.LoadMessage(movingUnderLineButtonPrompts,
      "underLineMovement");
00140                     // disable player rotation towards the camera and set player rotation 90 degrees
      in y axis to the line direction
00141                     camera3P.disableRotation = true;
00142                     var playerObject = transform.Find("Ch24_nonPBR");
00143                     playerObject.transform.rotation = Quaternion.LookRotation(lineDirection) *
      Quaternion.Euler(0, 90, 0);
00144 
00145                     // rotate minimap icon
00146                     minimapRotChangeY = transform.rotation.y - initialPlayerRotation.y > 0 ? -90 :
      +90;
00147                     minimapIcon.transform.rotation =
      Quaternion.Euler(minimapIcon.transform.rotation.eulerAngles.x,
      minimapIcon.transform.rotation.eulerAngles.y + minimapRotChangeY,
      minimapIcon.transform.rotation.eulerAngles.z);
00148                 }
00149             }
00150         }
00151     }
00152 
00157     private void OnCollisionExit(Collision collision)
00158     {
00159         if (collision.gameObject.CompareTag("movableLine"))
00160         {
00161             if (!isAboveLine)
00162             {
00163                 // rotate minimap icon back to default
00164                 minimapIcon.transform.rotation =
      Quaternion.Euler(minimapIcon.transform.rotation.eulerAngles.x,
      minimapIcon.transform.rotation.eulerAngles.y - minimapRotChangeY,
      minimapIcon.transform.rotation.eulerAngles.z);
00165             }
00166 
00167             playerRigidbody.useGravity = true;
00168             isMovingOnLine = false;
00169             playerMovement.enableAirMovement();
00170             camera3P.disableRotation = false;
00171         }
00172     }
00173 
```

```
00177     void Start()
00178     {
00179         playerRigidbody = GetComponent<Rigidbody>();
00180         buttonPromptsController = GetComponent<ScreenHints>();
00181         playerMovement = playerOrientation.parent.GetComponent<PlayerMovement>();
00182         minimapIcon = transform.Find("Ch24_nonPBR/MinimapPlayer");
00183     }
00184 
00188     void Update()
00189     {
00190         if (isMovingOnLine)
00191         {
00192             var angle = Vector3.Angle(playerOrientation.forward, lineDirection);
00193             direction = angle > 90 ? -1 : 1;
00194 
00195             if (Input.GetKey(KeyCode.W))
00196             {
00197                 moveTime = 0.0f;
00198                 // playerRigidbody.rotation = Quaternion.Slerp(playerRigidbody.rotation,
      Quaternion.LookRotation(lineDirection), Time.deltaTime);
00199                 playerRigidbody.velocity = direction * speed * lineDirection;
00200             }
00201             else if (Input.GetKey(KeyCode.S))
00202             {
00203                 moveTime = 0.0f;
00204                 //playerRigidbody.rotation = Quaternion.Slerp(playerRigidbody.rotation,
      Quaternion.LookRotation(lineDirection), Time.deltaTime);
00205                 playerRigidbody.velocity = direction * speed * -lineDirection;
00206             }
00207 
00208             if (Input.GetKeyUp(KeyCode.E))
00209             {
00210                 if (!isAboveLine)
00211                 {
00212                     playerRigidbody.useGravity = true;
00213                     isMovingOnLine = false;
00214                 }
00215             }
00216 
00217             // Check if the player is moving to avoid division by zero when normalizing the velocity
      vector
00218             if (playerRigidbody.velocity != Vector3.zero)
00219             {
00220                 // Calculate the direction from the velocity vector
00221                 Vector3 velocityDirection = playerRigidbody.velocity.normalized;
00222 
00223                 // Calculate the rotation that looks in the velocity direction with an up vector of
      Vector3.up
00224                 Quaternion targetRotation = Quaternion.LookRotation(velocityDirection, Vector3.up);
00225 
00226                 // Set the minimapIcon's rotation to match the target rotation, maintaining its
      current X and Z rotations
00227                 minimapIcon.transform.rotation =
      Quaternion.Euler(minimapIcon.transform.rotation.eulerAngles.x, targetRotation.eulerAngles.y,
      minimapIcon.transform.rotation.eulerAngles.z);
00228             }
00229 
00230             if (moveTime >= moveTimeMax)
00231             {
00232                 playerRigidbody.velocity = Vector3.zero;
00233             }
00234 
00235             moveTime += Time.deltaTime;
00236         }
00237     }
00238 
00242     public bool IsMovingOnLine()
00243     {
00244         return isMovingOnLine;
00245     }
00246 }
```

5.47 Assets/Scripts/MainMenu.cs File Reference

Classes

- class MainMenu

  This class manages the main menu interactions such as starting the game and quitting the application.

5.48 MainMenu.cs

Go to the documentation of this file.

```
00001 using System.Collections;
00002 using System.Collections.Generic;
00003 using UnityEngine;
00004 using UnityEngine.SceneManagement;
00005
00009 public class MainMenu : MonoBehaviour
00010 {
00014     public void PlayGame()
00015     {
00016         SceneManager.LoadScene(1);
00017     }
00018
00022     public void QuitGame()
00023     {
00024         Application.Quit();
00025     }
00026 }
```

5.49 Assets/Scripts/Manager.cs File Reference

Classes

- class Manager

  This class serves as a manager for general game functionalities.

5.50 Manager.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class Manager : MonoBehaviour
00007 {
00011     void Start()
00012     {
00013         // SaveSystem.Reset();
00014     }
00015
00019     void Update()
00020     {
00021
00022     }
00023 }
```

5.51 Assets/Scripts/MiniMapCamera.cs File Reference

Classes

- class MiniMapCamera

  The MiniMapCamera class handles the positioning and rotation of the minimap camera.

5.52 MiniMapCamera.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class MiniMapCamera : MonoBehaviour
00007 {
00009     [SerializeField]
00010     private Transform player;
00011
00015     void LateUpdate()
00016     {
00017         // Set the position of the minimap camera to be above the player
00018         transform.position = player.position + new Vector3(0, 15, 0);
00019
00020         // Set the rotation of the minimap camera to match the player's y rotation
00021         transform.rotation = Quaternion.Euler(90, player.eulerAngles.y, 0);
00022     }
00023 }
```

5.53 Assets/Scripts/MovableCameraController.cs File Reference

Classes

- class MovableCameraController

  The MovableCameraController class handles the movement and rotation of a movable camera in the scene. Movable camera is additional camera that is used to make different views of the scene for recording videos or taking screenshots.

5.54 MovableCameraController.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00007 public class MovableCameraController : MonoBehaviour
00008 {
00010     private Transform cameraTransform;
00011
00012     [Header("Movement")]
00014     public float movementSpeed = 10f;
00015
00017     public float fastMovementSpeed = 50f;
00018
00020     public float rotationSpeed = 5f;
00021
00023     public Transform objectToFollow;
00024
00028     void Start()
00029     {
00030         cameraTransform = transform.Find("Camera");
00031     }
00032
00033
00037     void Update()
00038     {
00039         if (objectToFollow == null)
00040         {
00041             // Check if left shift is held down and adjust movement speed accordingly
00042             float currentMovementSpeed = Input.GetKey(KeyCode.LeftShift) ? fastMovementSpeed :
      movementSpeed;
00043
00044             // Movement
00045             float horizontal = Input.GetAxis("Horizontal");
00046             float vertical = Input.GetAxis("Vertical");
00047
00048             Vector3 movementDirection = (transform.forward * vertical + transform.right *
      horizontal).normalized;
00049             transform.position += movementDirection * currentMovementSpeed * Time.deltaTime;
00050
00051             // Rotation
```

```
00052             if (Input.GetMouseButton(1)) // Right mouse button
00053             {
00054                 float mouseX = Input.GetAxis("Mouse X");
00055                 float mouseY = Input.GetAxis("Mouse Y");
00056 
00057                 Vector3 rotation = transform.eulerAngles;
00058                 rotation.y += mouseX * rotationSpeed;
00059                 rotation.x -= mouseY * rotationSpeed;
00060 
00061                 transform.eulerAngles = rotation;
00062             }
00063         }
00064         else
00065         {
00066             transform.position = objectToFollow.position + new Vector3(0, 4, -5);
00067             transform.rotation = objectToFollow.rotation;
00068         }
00069     }
00070 }
```

5.55 Assets/Scripts/MultipleTags.cs File Reference

Classes

- class MultipleTags

  The MultipleTags class allows a GameObject to have multiple tags.

5.56 MultipleTags.cs

Go to the documentation of this file.

```
00001 using System.Collections.Generic;
00002 using UnityEngine;
00003 
00007 public class MultipleTags : MonoBehaviour
00008 {
00010     [SerializeField]
00011     private List<string> tags;
00012 
00018     public bool HasTag(string tag) => tags.Contains(tag);
00019 
00024     public void AddTag(string tag)
00025     {
00026         if (!tags.Contains(tag))
00027         {
00028             tags.Add(tag);
00029         }
00030     }
00031 
00036     public void RemoveTag(string tag)
00037     {
00038         if (tags.Contains(tag))
00039         {
00040             tags.Remove(tag);
00041         }
00042     }
00043 
00048     public List<string> GetTags() => tags;
00049 }
```

5.57 Assets/Scripts/NPCController.cs File Reference

Classes

- class NPCController

  The NPCController class handles the interaction between the player and the NPCs in the game.

5.58 NPCController.cs

Go to the documentation of this file.

```
00001 using System.Collections.Generic;
00002 using TMPro;
00003 using UnityEngine;
00004
00008 public class NPCController : MonoBehaviour
00009 {
00010     [SerializeField]
00012     private string npcName;
00013
00014     [SerializeField]
00016     private TextAsset fileWithDialogue;
00017
00018     [SerializeField]
00020     private float lineDisplayTimeSec = 2.0f;
00021
00022     [SerializeField]
00024     private TMP_Text canvasText;
00025
00027     public Animator animator;
00028
00030     private int currentLine = 0;
00031
00033     private List<string> dialogue = new List<string>();
00034
00036     public bool dialogueActive = false;
00037
00039     private bool playerInRange = false;
00040
00042     private float currentLineDisplayTime = 0.0f;
00043
00044     public SoundEffectManager soundEffectManager;
00045
00049     void Start()
00050     {
00051         animator = GetComponent<Animator>();
00052
00053         string[] lines = fileWithDialogue.text.Split('\n');
00054         foreach (string line in lines)
00055         {
00056             dialogue.Add(line);
00057         }
00058     }
00059
00063     void Update()
00064     {
00065         if (playerInRange && currentLine == 0 && Input.GetKeyDown(KeyCode.E))
00066         {
00067             dialogueActive = true;
00068             canvasText.text = $"{npcName}: {dialogue[currentLine]}";
00069             canvasText.gameObject.SetActive(true);
00070         }
00071
00072         if (dialogueActive)
00073         {
00074             if (currentLine < dialogue.Count)
00075             {
00076                 if (animator)
00077                 {
00078                     animator.SetBool("isTalking", true);
00079                 }
00080                 if (currentLineDisplayTime >= lineDisplayTimeSec)
00081                 {
00082                     currentLineDisplayTime = 0.0f;
00083                     canvasText.text = $"{npcName}: {dialogue[currentLine]}";
00084                     currentLine++;
00085                 }
00086                 else
00087                 {
00088                     currentLineDisplayTime += Time.deltaTime;
00089                 }
00090             }
00091             else if (currentLine == dialogue.Count && currentLineDisplayTime >= lineDisplayTimeSec)
00092             {
00093                 if (animator)
00094                 {
00095                     animator.SetBool("isTalking", false);
00096                 }
00097                 dialogueActive = false;
00098                 currentLine = 0;
00099                 canvasText.gameObject.SetActive(false);
00100             }
00101             else
```

```
00102             {
00103                 currentLineDisplayTime += Time.deltaTime;
00104             }
00105         }
00106     }
00107
00111     private void OnTriggerEnter(Collider other)
00112     {
00113         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00114         {
00115             if (soundEffectManager)
00116             {
00117                 soundEffectManager.npcController = this;
00118             }
00119
00120             playerInRange = true;
00121             canvasText.gameObject.SetActive(true);
00122             canvasText.text = "Press E to talk to " + npcName;
00123         }
00124     }
00125
00129     private void OnTriggerExit(Collider other)
00130     {
00131         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00132         {
00133             playerInRange = false;
00134             currentLine = 0;
00135             dialogueActive = false;
00136             canvasText.gameObject.SetActive(false);
00137             canvasText.text = "";
00138         }
00139     }
00140 }
```

5.59 Assets/Scripts/PlayerAnimationStateController.cs File Reference

Classes

- class PlayerAnimationStateController

  Controls the animation states of the player character based on various game conditions.

5.60 PlayerAnimationStateController.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class PlayerAnimationStateController : MonoBehaviour
00007 {
00008     Animator animator;
00009
00010     [SerializeField]
00011     public PlayerMovement playerMovement;
00012
00013     public LineMovement lineMovement;
00014     public advancedClimbing advancedClimbing;
00015     public Ziplining ziplining;
00016
00017     int isCrouchingHash;
00018     int velocityHash;
00019     int isJumpingHash;
00020     int isWalkingOnLineHash;
00021     int isWalkingUnderLineHash;
00022     int isWalkingUnderLineDirectionHash;
00023     int isHangingHash;
00024     int isZipLiningHash;
00025     int velocityFlatHash;
00026
00027     // Start is called before the first frame update
00028     void Start()
00029     {
00030         animator = GetComponent<Animator>();
00031         isCrouchingHash = Animator.StringToHash("isCrouching");
00032         velocityHash = Animator.StringToHash("Velocity");
```

```
00033         velocityFlatHash = Animator.StringToHash("xVelocity");
00034         isJumpingHash = Animator.StringToHash("isJumping");
00035         isWalkingOnLineHash = Animator.StringToHash("isWalkingOnLine");
00036         isWalkingUnderLineHash = Animator.StringToHash("isWalkingUnderLine");
00037         isWalkingUnderLineDirectionHash = Animator.StringToHash("isWalkingUnderLineDirection");
00038         isHangingHash = Animator.StringToHash("isHanging");
00039         isZipLiningHash = Animator.StringToHash("isZipLining");
00040     }
00041
00042     // Update is called once per frame
00043     void Update()
00044     {
00045         animator.SetFloat(velocityFlatHash, playerMovement.velocityFlat / 16);
00046         bool isCrouching = animator.GetBool(isCrouchingHash);
00047         bool isJumping = animator.GetBool(isJumpingHash);
00048         bool isWalkingOnLine = animator.GetBool(isWalkingOnLineHash);
00049         bool isWalkingUnderLine = animator.GetBool(isWalkingUnderLineHash);
00050         bool isWalkingUnderLineDirection = animator.GetBool(isWalkingUnderLineDirectionHash);
00051         bool isHanging = animator.GetBool(isHangingHash);
00052         bool isZipLining = animator.GetBool(isZipLiningHash);
00053
00054         if (playerMovement.touchGround || lineMovement.isMovingOnLine || advancedClimbing.canHandle ||
      ziplining.isZiplining)
00055         {
00056             animator.SetBool(isJumpingHash, false);
00057             if (playerMovement.crouching)
00058             {
00059                 if (!isCrouching)
00060                 {
00061                     animator.SetBool(isCrouchingHash, true);
00062                 }
00063             }
00064             else
00065             {
00066                 if (isCrouching)
00067                 {
00068                     animator.SetBool(isCrouchingHash, false);
00069                 }
00070             }
00071             if (playerMovement.velocity > 0.1)
00072             {
00073                 animator.SetFloat(velocityHash, playerMovement.velocity / 16);
00074             }
00075             else
00076             {
00077                 if (animator.GetFloat(velocityHash) != 0)
00078                 {
00079                     animator.SetFloat(velocityHash, 0f);
00080                 }
00081             }
00082             if (lineMovement.isMovingOnLine)
00083             {
00084                 if (lineMovement.isAboveLine)
00085                 {
00086                     if (!isWalkingOnLine)
00087                     {
00088                         animator.SetBool(isWalkingOnLineHash, true);
00089                     }
00090                 }
00091                 else
00092                 {
00093                     if (!isWalkingUnderLine)
00094                     {
00095                         animator.SetBool(isWalkingUnderLineHash, true);
00096                     }
00097                     if (lineMovement.direction == 1)
00098                     {
00099                         animator.SetBool(isWalkingUnderLineDirectionHash, true);
00100                     }
00101                     else
00102                     {
00103                         animator.SetBool(isWalkingUnderLineDirectionHash, false);
00104                     }
00105                 }
00106             }
00107             else
00108             {
00109                 if (isWalkingOnLine || isWalkingUnderLine)
00110                 {
00111                     animator.SetBool(isWalkingOnLineHash, false);
00112                     animator.SetBool(isWalkingUnderLineHash, false);
00113                 }
00114             }
00115             if (advancedClimbing.canHandle)
00116             {
00117                 animator.SetBool(isHangingHash, true);
00118             }
```

```
00119             else
00120             {
00121                 animator.SetBool(isHangingHash, false);
00122             }
00123             if (ziplining.isZiplining)
00124             {
00125                 if (!isZipLining)
00126                 {
00127                     animator.SetBool(isZipLiningHash, true);
00128                 }
00129             }
00130             else
00131             {
00132                 if (isZipLining)
00133                 {
00134                     animator.SetBool(isZipLiningHash, false);
00135                 }
00136             }
00137         }
00138         else
00139         {
00140             animator.SetBool(isJumpingHash, true);
00141             animator.SetBool(isHangingHash, false);
00142             animator.SetBool(isWalkingUnderLineHash, false);
00143         }
00144
00145         if (Input.GetMouseButtonDown(1))
00146         {
00147             animator.SetBool("isKicking", true);
00148         }
00149         else
00150         {
00151             animator.SetBool("isKicking", false);
00152         }
00153
00154         if (Input.GetKey(KeyCode.C))
00155         {
00156             animator.SetBool("isFliping", true);
00157         }
00158         else
00159         {
00160             animator.SetBool("isFliping", false);
00161         }
00162     }
00163 }
```

5.61 Assets/Scripts/PlayerData.cs File Reference

Classes

- class PlayerData

  Serializable class representing player data for saving and loading.

5.62 PlayerData.cs

Go to the documentation of this file.

```
00001 [System.Serializable]
00005 public class PlayerData
00006 {
00008     public int currentLevel;
00009
00011     public int[] points;
00012
00018     public PlayerData(int level, int[] points)
00019     {
00020         currentLevel = level;
00021         this.points = points;
00022     }
00023 }
```

5.63 Assets/Scripts/PlayerInLift.cs File Reference

Classes

- class PlayerInLift

  Manages the player's animation state when inside a lift, based on his movement.

5.64 PlayerInLift.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class PlayerInLift : MonoBehaviour
00007 {
00009     [SerializeField]
00010     private Animator playerAnimator;
00011
00013     [SerializeField]
00014     private Rigidbody player;
00015
00021     private void OnCollisionStay(Collision collision)
00022     {
00023         // Check if the collision is with the player
00024         if (collision.gameObject.layer == LayerMask.NameToLayer("Player"))
00025         {
00026             // If the player is moving, set 'inLift' to false, else to true
00027             if (player.velocity.x != 0 || player.velocity.z != 0)
00028             {
00029                 playerAnimator.SetBool("inLift", false);
00030             }
00031             else
00032             {
00033                 playerAnimator.SetBool("inLift", true);
00034             }
00035         }
00036     }
00037
00043     private void OnCollisionExit(Collision collision)
00044     {
00045         // Check if the collision is with the player
00046         if (collision.gameObject.layer == LayerMask.NameToLayer("Player"))
00047         {
00048             // Reset the 'inLift' animation state to false
00049             playerAnimator.SetBool("inLift", false);
00050         }
00051     }
00052 }
```

5.65 Assets/Scripts/PlayerMovement.cs File Reference

Classes

- class PlayerMovement

  Manages player movement including walking, sprinting, crouching, and jumping.

5.66 PlayerMovement.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using System;
00003
00007 public class PlayerMovement : MonoBehaviour
00008 {
```

```
00009     [Header("Objects Ref")]
00011     public Transform orientation;
00012
00014     public LayerMask isGround;
00015
00017     public CapsuleCollider capsuleCollider;
00018
00019     [Header("Variables")]
00021     public float walkSpeedMultiplier;
00022
00024     public float sprintSpeedMultiplier;
00025
00027     public float crouchSpeedMultiplier;
00028
00030     public float moveSpeedLimit;
00031
00033     public float groundDrag;
00034
00036     public float jumpForce;
00037
00039     public float jumpCooldown;
00040
00042     public float airMovementMultiplier = 0.8f;
00043
00045     public float velocity;
00046
00048     public float velocityFlat;
00049
00050     private bool airMovementActive = true;
00051
00052     [Header("Slope Handling")]
00054     public float maxSlopeAngle;
00055     private RaycastHit slopeHit;
00056
00057     [Header("private")]
00058     [SerializeField]
00060     float horizontalI;
00061
00062     [SerializeField]
00064     float verticalI;
00065
00066     [SerializeField]
00068     float playerHeight;
00069
00070     [SerializeField]
00072     public bool touchGround;
00073
00074     [SerializeField]
00076     bool readyToJump;
00077
00079     Vector3 moveDir;
00080
00082     Rigidbody playerRigidbody;
00083
00084     private float moveSpeedMultiplier;
00086     public bool sprinting = false;
00087
00089     public bool crouching = false;
00090
00091     [SerializeField]
00093     public bool dead = false;
00094
00095     private Quaternion initialRotation;
00096
00100     void Start()
00101     {
00102         readyToJump = true;
00103         playerRigidbody = GetComponent<Rigidbody>();
00104         moveSpeedMultiplier = walkSpeedMultiplier;
00105         Cursor.lockState = CursorLockMode.Locked;
00106         SaveSystem.initialize();
00107     }
00108
00112     void Update()
00113     {
00114         touchGround = Physics.SphereCast(transform.position, 0.3f, Vector3.down, out RaycastHit hit,
      1f, isGround);
00115
00116         if (!dead)
00117         {
00118             inputControl();
00119             speedLimit();
00120
00121             if (touchGround)
00122             {
00123                 playerRigidbody.drag = groundDrag;
00124             }
```

```
00125             else
00126             {
00127                 playerRigidbody.drag = 0;
00128             }
00129 
00130             velocity = playerRigidbody.velocity.magnitude;
00131             velocityFlat = Math.Abs(playerRigidbody.velocity.x) +
      Math.Abs(playerRigidbody.velocity.z);
00132         }
00133         else
00134         {
00135             horizontalI = 0;
00136             verticalI = 0;
00137             transform.rotation = initialRotation;
00138         }
00139     }
00140 
00144     private void FixedUpdate()
00145     {
00146         if (!dead)
00147         {
00148             if (touchGround)
00149             {
00150                 groundMovement();
00151             }
00152             else if (airMovementActive)
00153             {
00154                 airMovement();
00155             }
00156         }
00157     }
00158 
00162     private void inputControl()
00163     {
00164         horizontalI = Input.GetAxisRaw("Horizontal");
00165         verticalI = Input.GetAxisRaw("Vertical");
00166         if (Input.GetKeyDown(KeyCode.Space) && touchGround && readyToJump && !onSteepSlope())
00167         {
00168             readyToJump = false;
00169             Invoke(nameof(jump), 0.2f);
00170             Invoke(nameof(afterJump), jumpCooldown);
00171             touchGround = false;
00172         }
00173 
00174         if (Input.GetKey(KeyCode.LeftShift) && touchGround && !crouching)
00175         {
00176             sprinting = true;
00177             moveSpeedMultiplier = sprintSpeedMultiplier;
00178         }
00179         if (Input.GetKeyUp(KeyCode.LeftShift) && !crouching)
00180         {
00181             sprinting = false;
00182             moveSpeedMultiplier = walkSpeedMultiplier;
00183         }
00184 
00185         if (Input.GetKeyDown(KeyCode.LeftControl) && touchGround)
00186         {
00187             capsuleCollider.height -= 0.6f;
00188             capsuleCollider.center -= Vector3.up * 0.3f;
00189             playerRigidbody.AddForce(Vector3.down * 5f, ForceMode.Impulse);
00190             readyToJump = false;
00191             crouching = true;
00192             moveSpeedMultiplier = crouchSpeedMultiplier;
00193         }
00194         if (Input.GetKeyUp(KeyCode.LeftControl) && crouching)
00195         {
00196             capsuleCollider.height += 0.6f;
00197             capsuleCollider.center += Vector3.up * 0.3f;
00198             readyToJump = true;
00199             crouching = false;
00200             moveSpeedMultiplier = walkSpeedMultiplier;
00201         }
00202     }
00203 
00207     private void speedLimit()
00208     {
00209         Vector2 rbSpeed = new Vector2(playerRigidbody.velocity.x, playerRigidbody.velocity.z);
00210         if (rbSpeed.magnitude > moveSpeedLimit)
00211         {
00212             rbSpeed = rbSpeed.normalized * moveSpeedMultiplier;
00213             playerRigidbody.velocity = new Vector3(rbSpeed.x, playerRigidbody.velocity.y, rbSpeed.y);
00214         }
00215     }
00216 
00220     private void groundMovement()
00221     {
00222         moveDir = (orientation.forward * verticalI) + (orientation.right * horizontalI);
```

```
00223         if (moveDir != Vector3.zero && transform.parent != null)
00224         {
00225             transform.parent = null;
00226         }
00227 
00228         if (onSlope())
00229         {
00230             playerRigidbody.AddForce(getSlopeMoveDirection() * moveSpeedMultiplier * 0.75f,
      ForceMode.Force);
00231         }
00232         else if (onSteepSlope())
00233         {
00234             playerRigidbody.AddForce(getSteepSlopeSlideDirection() * moveSpeedMultiplier * 0.75f,
      ForceMode.Force);
00235             playerRigidbody.AddForce(moveDir.normalized * moveSpeedMultiplier * 0.5f,
      ForceMode.Force);
00236         }
00237         else
00238         {
00239             playerRigidbody.AddForce(moveDir.normalized * moveSpeedMultiplier, ForceMode.Force);
00240         }
00241     }
00242 
00246     private void airMovement()
00247     {
00248         moveDir = (orientation.forward * verticalI) + (orientation.right * horizontalI);
00249         playerRigidbody.AddForce(moveDir.normalized * moveSpeedMultiplier * airMovementMultiplier,
      ForceMode.Force);
00250     }
00251 
00255     private void jump()
00256     {
00257         if (transform.parent != null)
00258         {
00259             transform.parent = null;
00260         }
00261         playerRigidbody.AddForce(transform.up * jumpForce, ForceMode.Impulse);
00262     }
00263 
00267     private void afterJump()
00268     {
00269         readyToJump = true;
00270         if (!Input.GetKey(KeyCode.LeftShift))
00271         {
00272             sprinting = false;
00273             moveSpeedMultiplier = walkSpeedMultiplier;
00274         }
00275     }
00276 
00280     public void enableAirMovement()
00281     {
00282         airMovementActive = true;
00283     }
00284 
00288     public void disableAirMovement()
00289     {
00290         airMovementActive = false;
00291     }
00292 
00297     private bool onSlope()
00298     {
00299         if (Physics.Raycast(transform.position, Vector3.down, out slopeHit, (capsuleCollider.height *
      0.5f) + 0.3f))
00300         {
00301             float angle = Vector3.Angle(Vector3.up, slopeHit.normal);
00302             return angle < maxSlopeAngle && angle != 0;
00303         }
00304         return false;
00305     }
00306 
00311     private bool onSteepSlope()
00312     {
00313         if (Physics.Raycast(transform.position, Vector3.down, out slopeHit, (capsuleCollider.height *
      0.5f) + 0.3f))
00314         {
00315             float angle = Vector3.Angle(Vector3.up, slopeHit.normal);
00316             return angle > maxSlopeAngle && angle != 0;
00317         }
00318         return false;
00319     }
00320 
00325     private Vector3 getSlopeMoveDirection()
00326     {
00327         return Vector3.ProjectOnPlane(moveDir, slopeHit.normal).normalized;
00328     }
00329 
00334     private Vector3 getSteepSlopeSlideDirection()
```

```
00335     {
00336         return Vector3.Cross(Vector3.Cross(slopeHit.normal, Vector3.down),
      slopeHit.normal).normalized;
00337     }
00338 }
```

5.67 Assets/Scripts/Portal.cs File Reference

Classes

- class Portal

  Represents a portal that loads a new scene when triggered by a collider.

5.68 Portal.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002 using UnityEngine.SceneManagement;
00003
00007 public class Portal : MonoBehaviour
00008 {
00010     public string sceneName;
00011
00017     private void OnTriggerEnter(Collider other)
00018     {
00019         SceneManager.LoadScene(sceneName);
00020     }
00021 }
```

5.69 Assets/Scripts/ResetPlayerAtStart.cs File Reference

Classes

- class ResetPlayerAtStart

  The ResetPlayerAtStart class is responsible for resetting the player's state at the start of the game. It is needed for the player to be correctly moving when on moving platforms.

5.70 ResetPlayerAtStart.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00007 public class ResetPlayerAtStart : MonoBehaviour
00008 {
00012     void Start()
00013     {
00014         GameObject player = GameObject.Find("Player");
00015         player.SetActive(false);
00016         player.SetActive(true);
00017     }
00018 }
```

5.71 Assets/Scripts/SaveSystem.cs File Reference

Classes

- class SaveSystem

  A static class for handling saving and loading player data using binary serialization.

5.72 SaveSystem.cs

Go to the documentation of this file.

```
00001 using System.IO;
00002 using System.Runtime.Serialization.Formatters.Binary;
00003 using UnityEngine;
00004
00008 public static class SaveSystem
00009 {
00013     public static void Reset()
00014     {
00015         Save(1, new int[5] { 0, 0, 0, 0, 0 });
00016     }
00017
00023     public static void Save(int level, int[] points)
00024     {
00025         BinaryFormatter formatter = new BinaryFormatter();
00026         string path = Application.persistentDataPath + "tarzan.gk";
00027         FileStream fs = new FileStream(path, FileMode.Create);
00028
00029         PlayerData data = new PlayerData(level, points);
00030         formatter.Serialize(fs, data);
00031         fs.Close();
00032     }
00033
00038     public static PlayerData Load()
00039     {
00040         string path = Application.persistentDataPath + "tarzan.gk";
00041         if (File.Exists(path))
00042         {
00043             BinaryFormatter formatter = new BinaryFormatter();
00044             FileStream fs = new FileStream(path, FileMode.Open);
00045             PlayerData pd = formatter.Deserialize(fs) as PlayerData;
00046             fs.Close();
00047             return pd;
00048         }
00049         else
00050         {
00051             Debug.Log("Can't detect save file");
00052             return null;
00053         }
00054     }
00055
00059     public static void initialize()
00060     {
00061         PlayerData loaded = Load();
00062         if (loaded == null)
00063         {
00064             int[] pointsData = new int[5];
00065
00066             for (int i = 0; i < 5; i++)
00067             {
00068                 pointsData[i] = 0;
00069             }
00070
00071             Save(1, pointsData);
00072         }
00073     }
00074
00080     public static void updateLevel(int level, int points)
00081     {
00082         PlayerData loaded = Load();
00083         if (loaded == null)
00084         {
00085             int[] pointsData = new int[5];
00086
00087             for (int i = 0; i < 5; i++)
00088             {
00089                 if (i == level - 1)
00090                 {
```

```
00091                     pointsData[i] = points;
00092                 }
00093                 else
00094                 {
00095                     pointsData[i] = 0;
00096                 }
00097             }
00098
00099             Save(level, pointsData);
00100         }
00101         else
00102         {
00103             int currentLevel = (level > loaded.currentLevel) ? level : loaded.currentLevel;
00104
00105             int[] currentPoints = loaded.points;
00106
00107             for (int i = 0; i < 5; i++)
00108             {
00109                 if ((i + 2) == level)
00110                 {
00111                     Debug.Log("points updated on level: " + (level - 1));
00112                     Debug.Log("old: " + currentPoints[i]);
00113                     currentPoints[i] = currentPoints[i] > points ? currentPoints[i] : points;
00114                     Debug.Log("new: " + currentPoints[i]);
00115                 }
00116             }
00117
00118             Save(currentLevel, currentPoints);
00119         }
00120     }
00121 }
```

5.73 Assets/Scripts/ScreenHints.cs File Reference

Classes

- class ScreenHints

  The ScreenHints class handles the display of on-screen messages.

5.74 ScreenHints.cs

Go to the documentation of this file.

```
00001 using System.Collections.Generic;
00002 using TMPro;
00003
00004 using UnityEngine;
00005
00009 public class ScreenHints : MonoBehaviour
00010 {
00012     private string[] messages;
00013
00015     public double timeToDisplay = 3.0;
00016
00018     private double timeDisplaying = 0.0;
00019
00021     private bool messageShown = false;
00022
00024     private bool isDisplaying = false;
00025
00027     private HashSet<string> loadedMessages = new HashSet<string>();
00028
00030     [SerializeField]
00031     private TMP_Text canvasText;
00032
00036     void Update()
00037     {
00038         if (isDisplaying)
00039         {
00040             if (timeDisplaying >= timeToDisplay)
00041             {
00042                 isDisplaying = false;
00043                 timeDisplaying = 0.0;
00044                 canvasText.gameObject.SetActive(false);
```

```
00045             }
00046             else
00047             {
00048                 timeDisplaying += Time.deltaTime;
00049             }
00050
00051             if (!messageShown)
00052             {
00053                 messageShown = true;
00054                 canvasText.text = "";
00055                 foreach (string message in messages)
00056                 {
00057                     canvasText.text += message + "\n";
00058                 }
00059                 canvasText.gameObject.SetActive(true);
00060             }
00061         }
00062     }
00063
00069     public void LoadMessage(string[] messages, string name)
00070     {
00071         if (isDisplaying)
00072             return;
00073
00074         if (loadedMessages.Contains(name))
00075             return;
00076
00077         this.messages = messages;
00078         loadedMessages.Add(name);
00079         isDisplaying = true;
00080         messageShown = false;
00081     }
00082 }
```

5.75 Assets/Scripts/ShowStats.cs File Reference

Classes

- class ShowStats

  Displays level completion information when a player enters a trigger collider.

5.76 ShowStats.cs

Go to the documentation of this file.

```
00001 using TMPro;
00002 using UnityEngine;
00003
00007 public class ShowStats : MonoBehaviour
00008 {
00012     public TMP_Text levelInfo;
00013
00017     public int level;
00018
00022     public int maxPoints;
00023
00029     private void OnTriggerEnter(Collider other)
00030     {
00031         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00032         {
00033             levelInfo.gameObject.SetActive(true);
00034             int currentLevel = SaveSystem.Load().currentLevel;
00035             if (currentLevel > level)
00036             {
00037                 int points = SaveSystem.Load().points[level - 1];
00038                 levelInfo.text = "Level " + level + " completed with " + points + "/" + maxPoints;
00039             }
00040             else
00041             {
00042                 levelInfo.text = "Level " + level + " incomplete";
00043             }
00044         }
00045     }
00046
```

```
00052     private void OnTriggerExit(Collider other)
00053     {
00054         if (other.gameObject.layer == LayerMask.NameToLayer("Player"))
00055         {
00056             levelInfo.gameObject.SetActive(false);
00057         }
00058     }
00059 }
```

5.77 Assets/Scripts/SoundEffectManager.cs File Reference

Classes

- class SoundEffectManager

  Manages sound effects based on various game events and states.

5.78 SoundEffectManager.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class SoundEffectManager : MonoBehaviour
00007 {
00011     public Animator animator;
00012
00016     public checkPointsMenager checkpointsmenager;
00017
00021     public PlayerMovement playerMovement;
00022
00026     public NPCController npcController;
00027
00031     public GoToLastCheckpointOnMine goToLastCheckpointOnMine;
00032
00036     int lastState;
00037
00041     float oldVelocity;
00042
00046     bool letTalk = false;
00047
00051     public AudioSource source;
00052
00056     public AudioClip walking;
00057
00061     public AudioClip runing;
00062
00066     public AudioClip jumping;
00067
00071     public AudioClip hanging;
00072
00076     public AudioClip death;
00077
00081     public AudioClip talk;
00082
00086     public AudioClip explosion;
00087
00091     public float walkRunLimit = 10;
00092
00093     // Start is called before the first frame update
00094     void Start()
00095     {
00096         AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
00097         lastState = stateInfo.shortNameHash;
00098         oldVelocity = playerMovement.velocity;
00099     }
00100
00101     // Update is called once per frame
00102     void Update()
00103     {
00104         AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
00105         int stateHash = stateInfo.shortNameHash;
00106
00107         // Check if the animator state or player velocity has changed
```

```
00108         if (stateHash != lastState || (oldVelocity > walkRunLimit && playerMovement.velocity <
      walkRunLimit) || (oldVelocity < walkRunLimit && playerMovement.velocity > walkRunLimit && stateHash ==
      Animator.StringToHash("Move Blend Tree")))
00109         {
00110             oldVelocity = playerMovement.velocity;
00111             lastState = stateHash;
00112 
00113             // Handle specific sound effects based on animator states
00114             if (stateHash == Animator.StringToHash("Jumping Up"))
00115             {
00116                 source.clip = jumping;
00117                 source.PlayOneShot(jumping);
00118             }
00119 
00120             if (stateHash == Animator.StringToHash("Hanging Idle 1"))
00121             {
00122                 source.clip = hanging;
00123                 source.PlayOneShot(hanging);
00124             }
00125 
00126             if (stateHash == Animator.StringToHash("Move Blend Tree"))
00127             {
00128                 if (oldVelocity < walkRunLimit)
00129                 {
00130                     source.clip = walking;
00131                     source.Play();
00132                 }
00133                 else
00134                 {
00135                     source.clip = runing;
00136                     source.Play();
00137                 }
00138             }
00139 
00140             if (stateHash == Animator.StringToHash("Idle"))
00141             {
00142                 source.clip = null;
00143             }
00144         }
00145 
00146         // Checkpoint manager handling
00147         if (checkpointsmenager)
00148         {
00149             if (checkpointsmenager.trigger)
00150             {
00151                 checkpointsmenager.trigger = false;
00152                 source.clip = death;
00153                 source.PlayOneShot(death);
00154             }
00155         }
00156 
00157         // NPC dialogue handling
00158         if (npcController)
00159         {
00160             if (npcController.dialogueActive)
00161             {
00162                 source.clip = talk;
00163                 if (!letTalk)
00164                 {
00165                     letTalk = true;
00166                     source.Play();
00167                 }
00168             }
00169             else
00170             {
00171                 letTalk = false;
00172             }
00173         }
00174 
00175         // Handling explosion sound effect
00176         if (goToLastCheckpointOnMine)
00177         {
00178             if (goToLastCheckpointOnMine.exploding)
00179             {
00180                 goToLastCheckpointOnMine.exploding = false;
00181                 source.PlayOneShot(explosion);
00182             }
00183         }
00184     }
00185 }
```

5.79 Assets/Scripts/StickyPlatform.cs File Reference

Classes

- class StickyPlatform

  The StickyPlatform class handles the interaction between the player and sticky platforms in the game. When the player is on a sticky platform, he become a child of the platform, moving with it. When the player stops colliding with the platform, he are no longer a child of the platform.

5.80 StickyPlatform.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00008 public class StickyPlatform : MonoBehaviour
00009 {
00011     private GameObject objectToStick;
00012
00016     private void Start()
00017     {
00018         objectToStick = GameObject.Find("Player");
00019     }
00020
00025     private void OnCollisionStay(Collision collision)
00026     {
00027         if (collision.gameObject == objectToStick)
00028         {
00029             objectToStick.transform.SetParent(transform);
00030         }
00031     }
00032
00037     private void OnCollisionExit(Collision collision)
00038     {
00039         if (collision.gameObject == objectToStick)
00040         {
00041             objectToStick.transform.SetParent(null);
00042         }
00043     }
00044 }
```

5.81 Assets/Scripts/VelocityText.cs File Reference

Classes

- class VelocityText

  Displays the current player velocity using TextMeshProUGUI.

5.82 VelocityText.cs

Go to the documentation of this file.

```
00001 using TMPro;
00002 using UnityEngine;
00003
00007 public class VelocityText : MonoBehaviour
00008 {
00012     public GameObject textMeshProVelocity;
00013
00017     public float playerVelocity;
00018
00022     TextMeshProUGUI textMeshProVelocityText;
00023
00024     [SerializeField]
```

```
00025     PlayerMovement playerMovement;
00026
00027     // Start is called before the first frame update
00028     void Start()
00029     {
00030         textMeshProVelocityText = textMeshProVelocity.GetComponent<TextMeshProUGUI>();
00031     }
00032
00033     // Update is called once per frame
00034     void Update()
00035     {
00036         playerVelocity = playerMovement.velocity;
00037         textMeshProVelocityText.text = "Velocity: " + playerVelocity.ToString("F2");
00038     }
00039 }
```

5.83 Assets/Scripts/Wallrunning.cs File Reference

Classes

- class Wallrunning

  Enables wall running mechanics for the player character.

5.84 Wallrunning.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class Wallrunning : MonoBehaviour
00007 {
00011     [Header("Wallrunning")]
00012     public LayerMask whatIsWall;
00013
00017     public LayerMask whatIsGround;
00018
00022     public float wallRunForce;
00023
00027     public float maxWallRunTime;
00028
00032     private float horizontalInput;
00033
00037     private float verticalInput;
00038
00042     [Header("Detection")]
00043     public float wallCheckDistance;
00044
00048     public float minJumpHeight;
00049
00053     private RaycastHit leftWallhit;
00054
00058     private RaycastHit rightWallhit;
00059
00063     private bool wallLeft;
00064
00068     private bool wallRight;
00069
00073     [Header("References")]
00074     public Transform orientation;
00075
00079     public bool wallrunning;
00080
00084     private Rigidbody rb;
00085
00086     private void Start()
00087     {
00088         rb = GetComponent<Rigidbody>();
00089     }
00090
00091     private void Update()
00092     {
00093         wallRight = Physics.Raycast(transform.position, orientation.right, out rightWallhit,
      wallCheckDistance, whatIsWall);
```

```
00094         wallLeft = Physics.Raycast(transform.position, -orientation.right, out leftWallhit,
      wallCheckDistance, whatIsWall);
00095
00096         horizontalInput = Input.GetAxisRaw("Horizontal");
00097         verticalInput = Input.GetAxisRaw("Vertical");
00098
00099         if ((wallLeft || wallRight) && verticalInput > 0 && CanWallRun())
00100         {
00101             if (!wallrunning) wallrunning = true;
00102         }
00103         else
00104         {
00105             if (wallrunning) wallrunning = false;
00106         }
00107     }
00108
00109     private void FixedUpdate()
00110     {
00111         if (wallrunning) WallRunningMovement();
00112     }
00113
00118     private bool CanWallRun()
00119     {
00120         return !Physics.Raycast(transform.position, Vector3.down, minJumpHeight, whatIsGround);
00121     }
00122
00126     private void WallRunningMovement()
00127     {
00128         rb.useGravity = false;
00129         rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
00130
00131         Vector3 wallNormal = wallRight ? rightWallhit.normal : leftWallhit.normal;
00132
00133         Vector3 wallForward = Vector3.Cross(wallNormal, transform.up);
00134
00135         if ((orientation.forward - wallForward).magnitude > (orientation.forward -
      -wallForward).magnitude)
00136             wallForward = -wallForward;
00137
00138         rb.AddForce(wallForward * wallRunForce, ForceMode.Force);
00139
00140         if (!(wallLeft && horizontalInput > 0) && !(wallRight && horizontalInput < 0))
00141             rb.AddForce(-wallNormal * 100, ForceMode.Force);
00142     }
00143 }
```

5.85 Assets/Scripts/WaypointsFollower.cs File Reference

Classes

- class WaypointsFollower

  The WaypointsFollower class handles the movement of an object along a set of waypoints.

5.86 WaypointsFollower.cs

Go to the documentation of this file.

```
00001 using UnityEngine;
00002
00006 public class WaypointsFollower : MonoBehaviour
00007 {
00009     [SerializeField]
00010     private float movingSpeed;
00011
00013     [SerializeField]
00014     private Transform[] waypoints;
00015
00017     private int nextWaypoint = 0;
00018
00020     [SerializeField]
00021     private bool rotateTowardsWaypoint = false;
00022
00026     private void Start()
00027     {
```

```
00028         if (GetComponent<Rigidbody>() != null)
00029         {
00030             movingSpeed *= 4;
00031         }
00032     }
00033
00037     private void Update()
00038     {
00039         transform.position = Vector3.MoveTowards(transform.position, waypoints[nextWaypoint].position,
      movingSpeed * Time.deltaTime);
00040         if (Vector3.Distance(transform.position, waypoints[nextWaypoint].position) < 0.1f)
00041         {
00042             nextWaypoint++;
00043         }
00044         if (nextWaypoint >= waypoints.Length)
00045         {
00046             nextWaypoint = 0;
00047         }
00048
00049         if (rotateTowardsWaypoint)
00050         {
00051             // set the object's rotation to look at the next waypoint
00052             transform.LookAt(waypoints[nextWaypoint]);
00053         }
00054     }
00055
00059     private void OnDrawGizmosSelected()
00060     {
00061         Gizmos.color = Color.blue;
00062         for (int i = 0; i < waypoints.Length - 1; i++)
00063         {
00064             Gizmos.DrawLine(waypoints[i].position, waypoints[i + 1].position);
00065         }
00066         Gizmos.DrawLine(waypoints[waypoints.Length - 1].position, waypoints[0].position);
00067     }
00068 }
```

5.87 Assets/Scripts/Ziplining.cs File Reference

Classes

- class Ziplining

  The Ziplining class handles the player's interaction with ziplines in the game.

5.88 Ziplining.cs

Go to the documentation of this file.

```
00001 using System.IO.Pipes;
00002 using UnityEngine;
00003
00007 public class Ziplining : MonoBehaviour
00008 {
00009     [Header("Ziplining Parameters")]
00010     [SerializeField]
00012     private float speed = 3.0f;
00013
00014     [Header("References")]
00015     [SerializeField]
00017     public Camera3P camera3P;
00018
00020     private Rigidbody playerRigidbody;
00021
00023     public bool isZiplining = false;
00024
00026     private Vector3 startPoint;
00027
00029     private Vector3 endPoint;
00030
00032     private Vector3 endLine;
00033
00035     private PlayerMovement playerMovement;
00036
00038     private Vector3 lineDirection;
```

```
00039
00041     private readonly string[] zipliningButtonsPrompts =
00042     {
00043         "press <sprite name=\"E\"> to let go"
00044     };
00045
00047     private ScreenHints buttonPromptsController;
00048
00052     void Start()
00053     {
00054         playerRigidbody = GetComponent<Rigidbody>();
00055         buttonPromptsController = GetComponent<ScreenHints>();
00056         playerMovement = playerRigidbody.GetComponent<PlayerMovement>();
00057     }
00058
00062     private void calculateLineDirection(Collision collision)
00063     {
00064         CapsuleCollider lineCollider = collision.gameObject.GetComponent<CapsuleCollider>();
00065
00066         // Get the line's length (subtract the diameter of the end cap spheres)
00067         float lineLength = lineCollider.height - (lineCollider.radius * 2);
00068
00069         // Calculate the start and end points
00070         switch (lineCollider.direction)
00071         {
00072             case 0: // X-axis
00073                 startPoint = collision.transform.position - (collision.transform.right * lineLength /
    2);
00074                 endPoint = collision.transform.position + (collision.transform.right * lineLength /
    2);
00075                 break;
00076             case 1: // Y-axis
00077                 startPoint = collision.transform.position - (collision.transform.up * lineLength / 2);
00078                 endPoint = collision.transform.position + (collision.transform.up * lineLength / 2);
00079                 break;
00080             case 2: // Z-axis
00081                 startPoint = collision.transform.position - (collision.transform.forward * lineLength
    / 2);
00082                 endPoint = collision.transform.position + (collision.transform.forward * lineLength /
    2);
00083                 break;
00084         }
00085
00086         // Check if end point is below start point
00087         if (endPoint.y > startPoint.y)
00088         {
00089             Vector3 temp = startPoint;
00090             startPoint = endPoint;
00091             endPoint = temp;
00092         }
00093
00094         // Calculate the direction of the line
00095         lineDirection = (endPoint - startPoint).normalized;
00096     }
00097
00102     private void OnCollisionEnter(Collision collision)
00103     {
00104         if (collision.gameObject.CompareTag("zippableLine"))
00105         {
00106             if (collision.contacts.Length > 0)
00107             {
00108                 playerMovement.disableAirMovement();
00109                 ContactPoint contact = collision.GetContact(0);
00110                 isZiplining = Vector3.Dot(contact.normal, Vector3.up) > 0.9f;
00111                 playerRigidbody.useGravity = false;
00112                 playerRigidbody.drag = 0f;
00113                 isZiplining = true;
00114                 endLine = collision.gameObject.GetComponent<MeshRenderer>().bounds.min;
00115                 calculateLineDirection(collision);
00116                 buttonPromptsController.LoadMessage(zipliningButtonsPrompts, "ziplining");
00117
00118                 // rotate player object to face the line direction
00119                 var playerObject = transform.Find("Ch24_nonPBR");
00120                 playerObject.transform.rotation = Quaternion.LookRotation(lineDirection);
00121
00122                 camera3P.disableRotation = true;
00123             }
00124         }
00125     }
00126
00130     private void OnCollisionExit(Collision collision)
00131     {
00132         if (collision.gameObject.CompareTag("zippableLine"))
00133         {
00134             playerRigidbody.useGravity = true;
00135             isZiplining = false;
00136             playerMovement.enableAirMovement();
```

```
00137
00138             camera3P.disableRotation = false;
00139             transform.Find("Ch24_nonPBR").transform.rotation = Quaternion.Euler(0,
      transform.rotation.eulerAngles.y, transform.rotation.eulerAngles.z);
00140         }
00141     }
00142
00146     void Update()
00147     {
00148         if (isZiplining)
00149         {
00150             Vector3 direction = (endPoint - startPoint).normalized;
00151             playerRigidbody.velocity = speed * direction;
00152
00153             if (Vector3.Distance(transform.position, endLine) < 1.6f)
00154             {
00155                 playerRigidbody.useGravity = true;
00156                 isZiplining = false;
00157             }
00158
00159             if (Input.GetKeyUp(KeyCode.E))
00160             {
00161                 playerRigidbody.useGravity = true;
00162                 isZiplining = false;
00163             }
00164         }
00165     }
00166
00170     public bool IsZiplining()
00171     {
00172         return isZiplining;
00173     }
00174 }
```

Index