Anfaengerpraktikum

v.0.1.0

Generated by Doxygen 1.12.0

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:

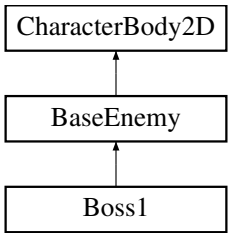
Chapter 4

Class Documentation

4.1 BaseEnemy Class Reference

Klasse für einen einfachen Gegner.

Inheritance diagram for BaseEnemy:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- override void _Process (double DeltaTime)

  Physikalische Prozesse werden in jedem Frame ausgeführt. Berechnet Gravitation und Bewegung.
- void OnDetectionBodyEntered (Node2D body)

  Detektiert den Spieler wenn er den Erkennungsbereich betritt.
- void OnPursuingRadiusBodyExited (Node2D body)

  Detektiert wenn der Spieler den Verfolgungsbereich verlässt.
- void OnHitboxAreaEntered (Area2D area)

  Detektiert wenn ein Objekt die Hitbox des Gegners betritt. (z.B.: Schwert des Spielers)
- void OnSwordHitBoxBodyEntered (Node2D body)

  Detektiert ob der Spieler in Schlagreichweite ist.
- void TakeDamage (Damage DMG)

  Verarbeitet zugefügten Schaden.
- bool IsDead ()

  Gibt boolean Dead zurück.
- void Die ()

  Wird aufgerufen wenn der Gegner stirbt.
- void Respawn ()

  Wird aufgerufen wenn der Gegner respawnt.

Public Attributes

- bool Dead = false
- float Armor = 20f
- float CurrentHealthPoints
- double ReturnToStart
- bool Pursuing = false
- Vector2 TargetPosition = Vector2.Inf
- Vector2 StartPosition
- AnimatedSprite2D Sprite
- CollisionShape2D MainCollision

Protected Member Functions

- virtual void UpdateAnimation ()

  Aktualisiert die Animationen des Gegners.

Protected Attributes

- float Damage = 20f
- bool Respawnable = true
- float MaxHealthPoints = 100f
- float MaxStamina = 1f
- float Speed = 10
- int SinAmount = 10
- double ReturnToStartAfter = 5
- float CurrentStamina
- Node2D CurrentTarget = null
- bool StartRotation = false
- bool AlreadyHit = false
- CollisionPolygon2D CollisionPolygon
- Area2D SwordHitbox
- RayCast2D FrontCollisionRayCast
- RayCast2D LineOfSight
- RayCast2D LeftFallProtection
- RayCast2D RightFallProtection
- TextureProgressBar HealthBar
- Player Player

Properties

- uint Id = 0 `[get, set]`

Private Types

- enum State { IDLE , WALK , ATTACK , TAKE_HIT }

Private Member Functions

- void HandleMovement (double DeltaTime)

  Verarbeitet die Bewegung des Gegners.
- void CheckPlayerHit ()

  Überprüft ob der Spieler sich, während eines Angriffes in Reichweite befindet und fügt diesem dann gegebenenfalls Schaden zu.
- bool CheckLineOfSight (Node2D body)

  Überprüft die direkte Sichtlinie zu einem Objekt.
- void FlipRotation ()

  Spiegelt die Orientierung aller zu dem Gegner gehörender Nodes.
- void SetRotation (bool Rotation)

  Setzt Orientierung aller zu dem Gegner gehörender Nodes.
- bool IsCloseTo (float Value1, float Value2, float Delta)

  Überprüft, ob zwei Werte in einer Delta-Umgebung zueinander liegen.

Private Attributes

- State AnimationState = State.IDLE

4.1.1 Detailed Description

Klasse für einen einfachen Gegner.

Definition at line 7 of file BaseEnemy.cs.

4.1.2 Member Enumeration Documentation

4.1.2.1 State

```
enum BaseEnemy.State  [private]
```

Enumerator

IDLE	
WALK	
ATTACK	
TAKE_HIT	

Definition at line 10 of file BaseEnemy.cs.

```
00010                       {
00011         IDLE, WALK, ATTACK, TAKE_HIT
00012     }
```

4.1.3 Member Function Documentation

4.1.3.1 _Process()

```
override void BaseEnemy._Process (
            double DeltaTime)  [inline]
```

Physikalische Prozesse werden in jedem Frame ausgeführt. Berechnet Gravitation und Bewegung.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 91 of file BaseEnemy.cs.

```
00092     {
00093         HandleMovement(DeltaTime);
00094         if(CurrentStamina < MaxStamina){
00095             CurrentStamina += (float) DeltaTime;
00096             Velocity = Velocity * 0.8f;
00097         }
00098         if (!IsOnFloor() && !Dead) {
00099             Velocity += GetGravity() * (float)DeltaTime;
00100         }
00101         UpdateAnimation();
00102         MoveAndSlide();
00103         CheckPlayerHit();
00104     }
```

4.1.3.2 _Ready()

```
override void BaseEnemy._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 64 of file BaseEnemy.cs.

```
00065     {
00066         Sprite = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
00067         CollisionPolygon = GetNode<CollisionPolygon2D>("detection/CollisionPolygon2D");
00068         SwordHitbox = GetNode<Area2D>("AnimatedSprite2D/SwordHitBox");
00069         MainCollision = GetNode<CollisionShape2D>("MainCollision");
00070         FrontCollisionRayCast = GetNode<RayCast2D>("FrontCollisionRayCast");
00071         LineOfSight = GetNode<RayCast2D>("LineOfSight");
00072         LeftFallProtection = GetNode<RayCast2D>("LeftFallProtection");
00073         RightFallProtection = GetNode<RayCast2D>("RightFallProtection");
00074         HealthBar = GetNode<TextureProgressBar>("HealthBar");
00075         Player = GetNode<Player>("../../Player");
00076 
00077         CurrentHealthPoints = MaxHealthPoints;
00078         CurrentStamina = MaxStamina;
00079         ReturnToStart = ReturnToStartAfter;
00080         StartPosition = Position;
00081         StartRotation = Sprite.FlipH;
00082 
00083         HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
00084     }
```

4.1.3.3 CheckLineOfSight()

```
bool BaseEnemy.CheckLineOfSight (
            Node2D body)  [inline], [private]
```

Überprüft die direkte Sichtlinie zu einem Objekt.

Parameters

body	Objekt das überprüft werden soll.

Returns

bool Ergebnis der Abfrage.

Definition at line 336 of file BaseEnemy.cs.

```
00336                                         {
00337         Vector2 offset = Vector2.Zero;
00338         offset.Y = -14;
00339         LineOfSight.TargetPosition = body.Position + offset - (Position + LineOfSight.Position);
00340         if(LineOfSight.IsColliding()){
00341             return LineOfSight.GetCollider() == body;
00342         }
00343         return true;
00344     }
```

4.1.3.4 CheckPlayerHit()

```
void BaseEnemy.CheckPlayerHit ()  [inline], [private]
```

Überprüft ob der Spieler sich, während eines Angriffes in Reichweite befindet und fügt diesem dann gegebenenfalls Schaden zu.

Definition at line 274 of file BaseEnemy.cs.

```
00274                                {
00275         if(Dead) return;
00276         if(Sprite.Animation != "attack"){
00277             AlreadyHit = false;
00278             if(Sprite.Animation == "take_hit" || CurrentStamina < MaxStamina) return;
00279             Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
00280             foreach(Node2D Body in Bodies){
00281                 if(Body == Player){
00282                     Sprite.Play("attack");
00283                 }
00284             }
00285             return;
00286         }
00287         if(AlreadyHit) return;
00288         if(Sprite.Frame >= 6){
00289             CurrentStamina = 0;
00290             Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
00291             foreach(Node2D Body in Bodies){
00292                 if(Body == Player){
00293                     Player.TakeDamage(new Damage(Damage, 0f, Vector2.Zero, this));
00294                     AlreadyHit = true;
00295                     break;
00296                 }
00297             }
00298         }
00299
00300     }
```

4.1.3.5 Die()

```
void BaseEnemy.Die ()  [inline]
```

Wird aufgerufen wenn der Gegner stirbt.

Definition at line 305 of file BaseEnemy.cs.

```
00305                          {
00306         Dead = true;
00307         Velocity = Vector2.Zero;
00308         MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, true);
00309
00310         Sprite.Play("death");
00311         HealthBar.SetVisible(false);
00312         if (Player != null) {
00313             Player.SetSinAmount(PlayerStats.Instance.GetSinAmount() + SinAmount);
00314         }
00315
00316     }
```

4.1.3.6 FlipRotation()

```
void BaseEnemy.FlipRotation ()  [inline], [private]
```

Spiegelt die Orientierung aller zu dem Gegner gehörender Nodes.

Definition at line 349 of file BaseEnemy.cs.

```
00349                               {
00350         Sprite.FlipH = !Sprite.FlipH;
00351         CollisionPolygon.RotationDegrees = Math.Abs(CollisionPolygon.RotationDegrees -180);
00352         SwordHitbox.RotationDegrees = Math.Abs(SwordHitbox.RotationDegrees -180);
00353         FrontCollisionRayCast.RotationDegrees = Math.Abs(FrontCollisionRayCast.RotationDegrees - 180);
00354     }
```

4.1.3.7 HandleMovement()

```
void BaseEnemy.HandleMovement (
            double DeltaTime)  [inline], [private]
```

Verarbeitet die Bewegung des Gegners.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 150 of file BaseEnemy.cs.

```
00150                                                   {
00151         if(Dead) return;
00152         if((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying()){
00153             Velocity = Vector2.Zero;
00154             return;
00155         }
00156         if(Pursuing){
00157             AnimationState = State.WALK;
00158             TargetPosition = CurrentTarget.Position;
00159             if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
00160                 AnimationState = State.IDLE;
00161                 Velocity = Vector2.Zero;
00162                 return;
00163             }
00164             ReturnToStart = ReturnToStartAfter;
00165         } else if(ReturnToStart >= 0){
00166             AnimationState = State.IDLE;
00167             ReturnToStart -= DeltaTime;
00168             TargetPosition = Vector2.Inf;
00169         } else if(!IsCloseTo(Position.X, StartPosition.X, 0.1f)){
00170             AnimationState = State.WALK;
00171             TargetPosition = StartPosition;
00172         }
00173 
00174         if(TargetPosition != Vector2.Inf){
00175 
00176             if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
00177                 AnimationState = State.IDLE;
00178                 Velocity = Vector2.Zero;
00179                 if(TargetPosition == StartPosition && Sprite.FlipH != StartRotation){
00180                     FlipRotation();
00181                 }
00182                 TargetPosition = Vector2.Inf;
00183                 return;
00184             }
00185 
00186             if(TargetPosition.X > Position.X){
00187                 SetRotation(true);
00188                 if(!FrontCollisionRayCast.IsColliding()){
00189                     Vector2 velocity = Vector2.Zero;
00190                     velocity.X = Speed;
00191                     Velocity = velocity;
00192                 }
00193             } else {
00194                 SetRotation(false);
00195                 if(!FrontCollisionRayCast.IsColliding()){
00196                     Vector2 velocity = Vector2.Zero;
00197                     velocity.X = -Speed;
00198                     Velocity = velocity;
00199                 }
00200             }
00201 
00202             if((!RightFallProtection.IsColliding() && !Sprite.FlipH) ||
      (!LeftFallProtection.IsColliding() && Sprite.FlipH)){
00203                 Velocity = Vector2.Zero;
00204             }
00205 
00206         } else {
00207             Velocity = Vector2.Zero;
00208             AnimationState = State.IDLE;
00209         }
00210     }
```

4.1.3.8 IsCloseTo()

```
bool BaseEnemy.IsCloseTo (
            float Value1,
            float Value2,
            float Delta)  [inline], [private]
```

Überprüft, ob zwei Werte in einer Delta-Umgebung zueinander liegen.

Parameters

float	Wert1
float	Wert2
float	Delta

Returns

bool Ergebnis

Definition at line 380 of file BaseEnemy.cs.

```
00380                                                                   {
00381         return Value1 <= (Value2 + Delta) && Value1 >= (Value2 - Delta);
00382     }
```

4.1.3.9 IsDead()

```
bool BaseEnemy.IsDead ()  [inline]
```

Gibt boolean Dead zurück.

Returns

bool ob Gegner tot ist.

Definition at line 266 of file BaseEnemy.cs.

```
00266                       {
00267         return Dead;
00268     }
```

4.1.3.10 OnDetectionBodyEntered()

```
void BaseEnemy.OnDetectionBodyEntered (
             Node2D body)  [inline]
```

Detektiert den Spieler wenn er den Erkennungsbereich betritt.

Parameters

body	Objekt das den Bereich betritt.

Definition at line 110 of file BaseEnemy.cs.

```
00110                                                   {
00111         if(CheckLineOfSight(body)){
00112             Pursuing = true;
00113             CurrentTarget = body;
00114         }
00115     }
```

4.1.3.11 OnHitboxAreaEntered()

```
void BaseEnemy.OnHitboxAreaEntered (
             Area2D area)  [inline]
```

Detektiert wenn ein Objekt die Hitbox des Gegners betritt. (z.B.: Schwert des Spielers)

Parameters

area	Objekt das den Bereich betritt.

Definition at line 132 of file BaseEnemy.cs.

```
00132                                                {
00133         Player Player1 = (Player) area.GetParent().GetParent();
00134         TakeDamage(Player1.GetDamage());
00135     }
```

4.1.3.12 OnPursuingRadiusBodyExited()

```
void BaseEnemy.OnPursuingRadiusBodyExited (
             Node2D body)  [inline]
```

Detektiert wenn der Spieler den Verfolgungsbereich verlässt.

Parameters

body	Objekt das den Bereich verlässt.

Definition at line 121 of file BaseEnemy.cs.

```
00121                                                  {
00122         if(body == CurrentTarget){
00123             Pursuing = false;
00124             CurrentTarget = null;
00125         }
00126     }
```

4.1.3.13 OnSwordHitBoxBodyEntered()

```
void BaseEnemy.OnSwordHitBoxBodyEntered (
             Node2D body)  [inline]
```

Detektiert ob der Spieler in Schlagreichweite ist.

Parameters

body	Objekt das den Bereich betritt.

Definition at line 141 of file BaseEnemy.cs.

```
00141                                                  {
00142         if(Dead) return;
00143         Sprite.Play("attack");
00144     }
```

4.1.3.14 Respawn()

```
void BaseEnemy.Respawn ()  [inline]
```

Wird aufgerufen wenn der Gegner respawnt.

Definition at line 321 of file BaseEnemy.cs.

```
00322     {
00323         Dead = false;
00324         CurrentHealthPoints = MaxHealthPoints;
00325         HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00326         MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, false);
00327         HealthBar.SetVisible(true);
00328         Sprite.Play("idle");
00329     }
```

4.1.3.15 SetRotation()

```
void BaseEnemy.SetRotation (
            bool Rotation)  [inline], [private]
```

Setzt Orientierung aller zu dem Gegner gehörender Nodes.

Parameters

Rotation	Die neue Orientierung.

Definition at line 360 of file BaseEnemy.cs.

```
00360                                           {
00361         Sprite.FlipH = Rotation ^ StartRotation; // XOR mit StartRotation
00362         if(Rotation){
00363             CollisionPolygon.RotationDegrees = 180;
00364             SwordHitbox.RotationDegrees = 180;
00365             FrontCollisionRayCast.RotationDegrees = 180;
00366         } else {
00367             CollisionPolygon.RotationDegrees = 0;
00368             SwordHitbox.RotationDegrees = 0;
00369             FrontCollisionRayCast.RotationDegrees = 0;
00370         }
00371     }
```

4.1.3.16 TakeDamage()

```
void BaseEnemy.TakeDamage (
            Damage DMG)  [inline]
```

Verarbeitet zugefügten Schaden.

Parameters

DMG	Schaden der zugefügt wird.

Definition at line 245 of file BaseEnemy.cs.

```
00245                                       {
00246         if(Dead) {
00247             return;
00248         }
00249         CurrentHealthPoints -= DMG.GetPhysicalDMG() * (1 - Armor / 100.0f) + DMG.GetTrueDMG();
00250         Position += DMG.GetPushAmount();
00251         if(CurrentHealthPoints <= 0){
00252             Die();
00253         } else {
00254             Sprite.Play("take_hit");
00255             if(DMG.GetSource() == Player){
00256                 Pursuing = true;
00257                 CurrentTarget = Player;
00258             }
00259         }
00260     }
```

4.1.3.17 UpdateAnimation()

```
virtual void BaseEnemy.UpdateAnimation ()  [inline], [protected], [virtual]
```

Aktualisiert die Animationen des Gegners.

Definition at line 216 of file BaseEnemy.cs.

```
00216                                           {
00217         if(Dead) return;
00218         if(!((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying())){
00219             switch(AnimationState){
00220                 case State.IDLE:
00221                     Sprite.Play("idle");
00222                     break;
00223 
00224                 case State.WALK:
00225                     Sprite.Play("walk");
00226                     break;
00227 
00228                 case State.ATTACK:
00229                     Sprite.Play("attack");
00230                     break;
00231 
00232                 case State.TAKE_HIT:
00233                     Sprite.Play("take_hit");
00234                     break;
00235             }
00236         }
00237         HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
00238 
00239     }
```

4.1.4 Member Data Documentation

4.1.4.1 AlreadyHit

```
bool BaseEnemy.AlreadyHit = false  [protected]
```

Definition at line 46 of file BaseEnemy.cs.

4.1.4.2 AnimationState

```
State BaseEnemy.AnimationState = State.IDLE  [private]
```

Definition at line 45 of file BaseEnemy.cs.

4.1.4.3 Armor

```
float BaseEnemy.Armor = 20f
```

Definition at line 24 of file BaseEnemy.cs.

4.1.4.4 CollisionPolygon

```
CollisionPolygon2D BaseEnemy.CollisionPolygon  [protected]
```

Definition at line 50 of file BaseEnemy.cs.

4.1.4.5 CurrentHealthPoints

```
float BaseEnemy.CurrentHealthPoints
```

Definition at line 37 of file BaseEnemy.cs.

4.1.4.6 CurrentStamina

```
float BaseEnemy.CurrentStamina  [protected]
```

Definition at line 38 of file BaseEnemy.cs.

4.1.4.7 CurrentTarget

```
Node2D BaseEnemy.CurrentTarget = null  [protected]
```

Definition at line 41 of file BaseEnemy.cs.

4.1.4.8 Damage

```
float BaseEnemy.Damage = 20f  [protected]
```

Definition at line 16 of file BaseEnemy.cs.

4.1.4.9 Dead

```
bool BaseEnemy.Dead = false
```

Definition at line 18 of file BaseEnemy.cs.

4.1.4.10 FrontCollisionRayCast

```
RayCast2D BaseEnemy.FrontCollisionRayCast  [protected]
```

Definition at line 53 of file BaseEnemy.cs.

4.1.4.11 HealthBar

```
TextureProgressBar BaseEnemy.HealthBar  [protected]
```

Definition at line 57 of file BaseEnemy.cs.

4.1.4.12 LeftFallProtection

```
RayCast2D BaseEnemy.LeftFallProtection  [protected]
```

Definition at line 55 of file BaseEnemy.cs.

4.1.4.13 LineOfSight

```
RayCast2D BaseEnemy.LineOfSight  [protected]
```

Definition at line 54 of file BaseEnemy.cs.

4.1.4.14 MainCollision

```
CollisionShape2D BaseEnemy.MainCollision
```

Definition at line 52 of file BaseEnemy.cs.

4.1.4.15 MaxHealthPoints

```
float BaseEnemy.MaxHealthPoints = 100f  [protected]
```

Definition at line 22 of file BaseEnemy.cs.

4.1.4.16 MaxStamina

```
float BaseEnemy.MaxStamina = 1f  [protected]
```

Definition at line 26 of file BaseEnemy.cs.

4.1.4.17 Player

```
Player BaseEnemy.Player  [protected]
```

Definition at line 58 of file BaseEnemy.cs.

4.1.4.18 Pursuing

```
bool BaseEnemy.Pursuing = false
```

Definition at line 40 of file BaseEnemy.cs.

4.1.4.19 Respawnable

```
bool BaseEnemy.Respawnable = true  [protected]
```

Definition at line 20 of file BaseEnemy.cs.

4.1.4.20 ReturnToStart

```
double BaseEnemy.ReturnToStart
```

Definition at line 39 of file BaseEnemy.cs.

4.1.4.21 ReturnToStartAfter

```
double BaseEnemy.ReturnToStartAfter = 5  [protected]
```

Definition at line 32 of file BaseEnemy.cs.

4.1.4.22 RightFallProtection

```
RayCast2D BaseEnemy.RightFallProtection  [protected]
```

Definition at line 56 of file BaseEnemy.cs.

4.1.4.23 SinAmount

```
int BaseEnemy.SinAmount = 10  [protected]
```

Definition at line 30 of file BaseEnemy.cs.

4.1.4.24 Speed

```
float BaseEnemy.Speed = 10  [protected]
```

Definition at line 28 of file BaseEnemy.cs.

4.1.4.25 Sprite

```
AnimatedSprite2D BaseEnemy.Sprite
```

Definition at line 49 of file BaseEnemy.cs.

4.1.4.26 StartPosition

```
Vector2 BaseEnemy.StartPosition
```

Definition at line 43 of file BaseEnemy.cs.

4.1.4.27 StartRotation

```
bool BaseEnemy.StartRotation = false  [protected]
```

Definition at line 44 of file BaseEnemy.cs.

4.1.4.28 SwordHitbox

```
Area2D BaseEnemy.SwordHitbox  [protected]
```

Definition at line 51 of file BaseEnemy.cs.

4.1.4.29 TargetPosition

```
Vector2 BaseEnemy.TargetPosition = Vector2.Inf
```

Definition at line 42 of file BaseEnemy.cs.

4.1.5 Property Documentation

4.1.5.1 Id

```
uint BaseEnemy.Id = 0  [get], [set]
```

Definition at line 34 of file BaseEnemy.cs.

```
00034 { get; set;} = 0;
```

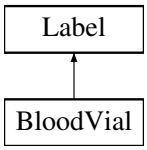
The documentation for this class was generated from the following file:

- BaseEnemy.cs

4.2 BloodVial Class Reference

Klasse für die Interaktion zum heilen.

Inheritance diagram for BloodVial:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- void UseBloodVial ()

  Versucht ein Bloodvial zu verwenden um den Spieler zu Heilen.
- void ResetUses ()

  Setzt die Anzahl der Bloodvials auf das Maximum.
- void AddMaxUses (int Amount)

  Verbessert die Maximale Anzahl an Bloodvials um die angegebene Anzahl.
- void LevelHealAmount ()

  Verbessert den HealAMount eines Bloodvials um 25.

4.2.1 Detailed Description

Klasse für die Interaktion zum heilen.

Definition at line 8 of file BloodVial.cs.

4.2.2 Member Function Documentation

4.2.2.1 _Ready()

```
override void BloodVial._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 14 of file BloodVial.cs.

```
00014                                {
00015         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00016     }
```

4.2.2.2 AddMaxUses()

```
void BloodVial.AddMaxUses (
             int Amount)  [inline]
```

Verbessert die Maximale Anzahl an Bloodvials um die angegebene Anzahl.

Parameters

int	Amount, um die MaxUses erhöht wird.

Definition at line 40 of file BloodVial.cs.

```
00040                                    {
00041         PlayerStats.Instance.SetBVMaxUses(PlayerStats.Instance.GetBVMaxUses() + Amount);
00042         ResetUses();
00043     }
```

4.2.2.3 LevelHealAmount()

```
void BloodVial.LevelHealAmount ()  [inline]
```

Verbessert den HealAMount eines Bloodvials um 25.

Definition at line 48 of file BloodVial.cs.

```
00048                                  {
00049         PlayerStats.Instance.SetBVHealAmount(PlayerStats.Instance.GetBVHealAmount() + 25);
00050     }
```

4.2.2.4 ResetUses()

```
void BloodVial.ResetUses ()  [inline]
```

Setzt die Anzahl der Bloodvials auf das Maximum.

Definition at line 31 of file BloodVial.cs.

```
00031                            {
00032         PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVMaxUses());
00033         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00034     }
```

4.2.2.5 UseBloodVial()

```
void BloodVial.UseBloodVial ()  [inline]
```

Versucht ein Bloodvial zu verwenden um den Spieler zu Heilen.

Definition at line 21 of file BloodVial.cs.

```
00021                           {
00022         if(PlayerStats.Instance.GetBVCurrentUses() <= 0) return;
00023         PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVCurrentUses() - 1);
00024         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00025         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() +
      PlayerStats.Instance.GetBVHealAmount());
00026     }
```

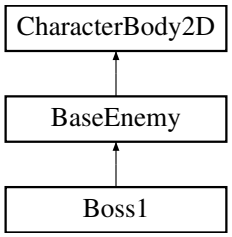
The documentation for this class was generated from the following file:

- BloodVial.cs

4.3 Boss1 Class Reference

Klasse für einen stärkeren Boss-Gegner, der von BaseEnemy erbt.

Inheritance diagram for Boss1:



Public Member Functions

- override void _Ready ()

  Überschreibt die _Ready-Methode von BaseEnemy.
- override void _Process (double DeltaTime)

  Überschreibt die _Process-Methode von BaseEnemy.
- void HandleRegeneration (double DeltaTime)

  Regeneriert die Gesundheit des Bosses, wenn er keinen Schaden nimmt.
- void StartGlowing ()

  Startet einen Leuchteffekt, wenn der Boss Schaden nimmt.

Public Member Functions inherited from BaseEnemy

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- override void _Process (double DeltaTime)

  Physikalische Prozesse werden in jedem Frame ausgeführt. Berechnet Gravitation und Bewegung.
- void OnDetectionBodyEntered (Node2D body)

  Detektiert den Spieler wenn er den Erkennungsbereich betritt.
- void OnPursuingRadiusBodyExited (Node2D body)

  Detektiert wenn der Spieler den Verfolgungsbereich verlässt.
- void OnHitboxAreaEntered (Area2D area)

  Detektiert wenn ein Objekt die Hitbox des Gegners betritt. (z.B.: Schwert des Spielers)
- void OnSwordHitBoxBodyEntered (Node2D body)

  Detektiert ob der Spieler in Schlagreichweite ist.
- void TakeDamage (Damage DMG)

  Verarbeitet zugefügten Schaden.
- bool IsDead ()

  Gibt boolean Dead zurück.
- void Die ()

  Wird aufgerufen wenn der Gegner stirbt.
- void Respawn ()

  Wird aufgerufen wenn der Gegner respawnt.

Public Attributes

- bool EnemiesRevived = false

Public Attributes inherited from BaseEnemy

- bool Dead = false
- float Armor = 20f
- float CurrentHealthPoints
- double ReturnToStart
- bool Pursuing = false
- Vector2 TargetPosition = Vector2.Inf
- Vector2 StartPosition
- AnimatedSprite2D Sprite
- CollisionShape2D MainCollision

Private Member Functions

- void ShowPopupMessage (string Message)

  Zeigt eine Popup-Nachricht an.
- void ReviveEnemies ()

  Lässt alle toten Feinde im Raum des Bosses wiederbeleben.

Private Attributes

- float RegenCooldown = 5.0f
- float RegenTimer = 0.0f
- float RegenAmount = 10.0f

Additional Inherited Members

Protected Member Functions inherited from BaseEnemy

- virtual void UpdateAnimation ()

  Aktualisiert die Animationen des Gegners.

Protected Attributes inherited from BaseEnemy

- float Damage = 20f
- bool Respawnable = true
- float MaxHealthPoints = 100f
- float MaxStamina = 1f
- float Speed = 10
- int SinAmount = 10
- double ReturnToStartAfter = 5
- float CurrentStamina
- Node2D CurrentTarget = null
- bool StartRotation = false
- bool AlreadyHit = false
- CollisionPolygon2D CollisionPolygon
- Area2D SwordHitbox
- RayCast2D FrontCollisionRayCast
- RayCast2D LineOfSight
- RayCast2D LeftFallProtection
- RayCast2D RightFallProtection
- TextureProgressBar HealthBar
- Player Player

Properties inherited from BaseEnemy

- uint Id = 0 `[get, set]`

4.3.1 Detailed Description

Klasse für einen stärkeren Boss-Gegner, der von BaseEnemy erbt.

Definition at line 7 of file Boss1.cs.

4.3.2 Member Function Documentation

4.3.2.1 _Process()

```
override void Boss1._Process (
            double DeltaTime)  [inline]
```

Überschreibt die _Process-Methode von BaseEnemy.

Parameters

DeltaTime	Die Zeit, die seit dem letzten Frame vergangen ist

Definition at line 36 of file Boss1.cs.

```
00036                                                {
00037         base._Process(DeltaTime);
00038
00039         if (CurrentHealthPoints <= MaxHealthPoints / 2 && !EnemiesRevived){
00040             StartGlowing();
00041             ReviveEnemies();
00042             EnemiesRevived = true;
00043             Armor = 60f; // Rüstung erhöhen
00044         }
00045
00046         HandleRegeneration(DeltaTime);
00047     }
```

4.3.2.2 _Ready()

```
override void Boss1._Ready ()  [inline]
```

Überschreibt die _Ready-Methode von BaseEnemy.

Definition at line 18 of file Boss1.cs.

```
00018                                    {
00019
00020         MaxHealthPoints = 400f;
00021         Damage = 50f;
00022         Armor = 30f;
00023         Speed = 10f;
00024         SinAmount = 100; // Bonuspunkte für Spieler beim Besiegen des Bosses
00025
00026         base._Ready();
00027
00028         CurrentHealthPoints = MaxHealthPoints;
00029         HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00030     }
```

4.3.2.3 HandleRegeneration()

```
void Boss1.HandleRegeneration (
             double DeltaTime)  [inline]
```

Regeneriert die Gesundheit des Bosses, wenn er keinen Schaden nimmt.

Parameters

DeltaTime	Die Zeit, die seit dem letzten Frame vergangen ist

Definition at line 53 of file Boss1.cs.

```
00053                                                {
00054         if (CurrentHealthPoints < MaxHealthPoints){
00055             RegenTimer += (float)DeltaTime;
00056
00057             if (RegenTimer >= RegenCooldown){
00058                 CurrentHealthPoints = Math.Min(CurrentHealthPoints + RegenAmount, MaxHealthPoints);
00059                 HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00060                 RegenTimer = 0.0f; // Timer zurücksetzen
00061             }
00062         }
00063     }
```

4.3.2.4 ReviveEnemies()

```
void Boss1.ReviveEnemies ()  [inline], [private]
```

Lässt alle toten Feinde im Raum des Bosses wiederbeleben.

Definition at line 108 of file Boss1.cs.

```
00109     {
00110         // Hole den Elternknoten (bossRoom)
00111         Node BossRoom = GetParent();
00112 
00113         // Iteriere durch alle Kinder von bossRoom
00114         foreach (Node Child in BossRoom.GetChildren()){
00115             if (Child is BaseEnemy BaseEnemy && BaseEnemy.IsDead()){
00116                 BaseEnemy.Respawn();
00117             }
00118         }
00119     }
```

4.3.2.5 ShowPopupMessage()

```
void Boss1.ShowPopupMessage (
             string Message)  [inline], [private]
```

Zeigt eine Popup-Nachricht an.

Parameters

Message	Die Nachricht, die angezeigt werden soll

Definition at line 80 of file Boss1.cs.

```
00080                                                 {
00081         Label popup = new Label();
00082         popup.Text = Message;
00083         popup.AddThemeColorOverride("font_color", new Color(1, 0, 0)); // Rot
00084         popup.Modulate = new Color(1, 1, 1, 0); // Start transparent
00085         popup.AutowrapMode = TextServer.AutowrapMode.Word;
00086         popup.SizeFlagsHorizontal = (Control.SizeFlags)(int)Control.SizeFlags.ExpandFill;
00087         popup.SizeFlagsVertical = (Control.SizeFlags)(int)Control.SizeFlags.ShrinkCenter;
00088         popup.HorizontalAlignment = HorizontalAlignment.Center;
00089         popup.VerticalAlignment = VerticalAlignment.Center;
00090 
00091 
00092         Vector2 bossGlobalPosition = GetGlobalTransformWithCanvas().Origin;
00093         popup.GlobalPosition = bossGlobalPosition + new Vector2(0, -100);
00094 
00095         CanvasLayer canvas = new CanvasLayer();
00096         AddChild(canvas);
00097         canvas.AddChild(popup);
00098 
00099         var tween = CreateTween();
00100         tween.TweenProperty(popup, "modulate:a", 1, 0.5f).From(0); // Einblenden
00101         tween.TweenProperty(popup, "modulate:a", 0, 0.5f).From(1).SetDelay(1.0f); // Ausblenden nach 1
      Sekunde
00102         tween.Connect("finished", new Callable(popup, "queue_free"));
00103     }
```

4.3.2.6 StartGlowing()

```
void Boss1.StartGlowing ()  [inline]
```

Startet einen Leuchteffekt, wenn der Boss Schaden nimmt.

Definition at line 68 of file Boss1.cs.

```
00068                                 {
00069         // Ändere die Modulationsfarbe des Sprites, um ein Leuchten zu simulieren
00070         if (Sprite != null){
00071             ShowPopupMessage("AHHHH!!!");
00072             Sprite.Modulate = new Color(1.0f, 0.84f, 0.0f, 1.0f); // Ein goldliche Leuchteffekt
00073         }
00074     }
```

4.3.3 Member Data Documentation

4.3.3.1 EnemiesRevived

```
bool Boss1.EnemiesRevived = false
```

Definition at line 9 of file Boss1.cs.

4.3.3.2 RegenAmount

```
float Boss1.RegenAmount = 10.0f  [private]
```

Definition at line 12 of file Boss1.cs.

4.3.3.3 RegenCooldown

```
float Boss1.RegenCooldown = 5.0f  [private]
```

Definition at line 10 of file Boss1.cs.

4.3.3.4 RegenTimer

```
float Boss1.RegenTimer = 0.0f  [private]
```

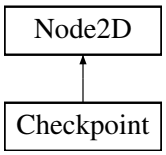
Definition at line 11 of file Boss1.cs.

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

- Boss1.cs

4.4 Checkpoint Class Reference

Inheritance diagram for Checkpoint:



Public Member Functions

- override void _Ready ()

Private Member Functions

- void OnPlayerBodyEntered (Node body)

Private Attributes

- Player Player

4.4.1 Detailed Description

Definition at line 4 of file Checkpoint.cs.

4.4.2 Member Function Documentation

4.4.2.1 _Ready()

```
override void Checkpoint._Ready ()  [inline]
```

Definition at line 13 of file Checkpoint.cs.

```
00014     {
00015         // Zugriff auf Player Node
00016         Player = GetNode<Player>("../Player");
00017     }
```

4.4.2.2 OnPlayerBodyEntered()

```
void Checkpoint.OnPlayerBodyEntered (
            Node body)  [inline], [private]
```

Prüfen ob der Körper, der den Checkpoint betritt, ein Player ist Wenn ja, dann wird der Checkpoint als Spawnpoint gesetzt

Definition at line 23 of file Checkpoint.cs.

```
00024     {
00025
00031         if (body is Player Player)
00032         {
00033             // Setzen des Spawnpoints
00034             PlayerStats PlayerStats = GetNode<PlayerStats>("/root/PlayerStats");
00035             PlayerStats.Instance.SetSpawnPoint(this.GlobalPosition);
00036             Player.MaxHeal();
00037             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetMaxStamina());
00038             Player.GetBloodVials().ResetUses();
00039             GD.Print("Spawnpoint des Players gesetzt auf: ", this.GlobalPosition);
00040
00041             PlayerStats.SetRespawnLevelTag(GetParent().Name);
00042             GD.Print("RespawnLevelTag des Players gesetzt auf: ", GetParent().Name);
00043             GD.Print(PlayerStats.Instance.GetRespawnLevelTag());
00044         }
00045
00046     }
```

4.4.3 Member Data Documentation

4.4.3.1 Player

```
Player Checkpoint.Player  [private]
```

Definition at line 8 of file Checkpoint.cs.

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

- Checkpoint.cs

4.5 Damage Class Reference

Repräsentiert den Schaden, der von Charakteren oder Gegnern verursacht wird. Beinhaltet physischen Schaden, wahren Schaden und den Rückstoßeffekt.

Public Member Functions

- Damage (float PhysicalDMG, float TrueDMG, Vector2 PushAmount, Node2D Source)
  Konstruktor für die Damage-Klasse.
- float GetPhysicalDMG ()
  Gibt den physischen Schaden zurück.
- float GetTrueDMG ()
  Gibt den wahren Schaden zurück.
- Vector2 GetPushAmount ()
  Gibt den Rückstoßvektor zurück.
- Node2D GetSource ()
  Gibt die Ursache zurück.

Private Attributes

- float PhysicalDMG
- float TrueDMG
- Vector2 PushAmount
- Node2D Source

4.5.1 Detailed Description

Repräsentiert den Schaden, der von Charakteren oder Gegnern verursacht wird. Beinhaltet physischen Schaden, wahren Schaden und den Rückstoßeffekt.

Definition at line 7 of file Damage.cs.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Damage()

```
Damage.Damage (
            float PhysicalDMG,
            float TrueDMG,
            Vector2 PushAmount,
            Node2D Source)  [inline]
```

Konstruktor für die Damage-Klasse.

Parameters

PhysicalDMG	Der physische Schaden.
TrueDMG	Der wahre Schaden.
PushAmount	Der Rückstoßvektor.

Definition at line 20 of file Damage.cs.

```
00020                                                                          {
00021         this.PhysicalDMG = PhysicalDMG;
00022         this.TrueDMG = TrueDMG;
00023         this.PushAmount = PushAmount;
00024         this.Source = Source;
00025     }
```

4.5.3 Member Function Documentation

4.5.3.1 GetPhysicalDMG()

`float Damage.GetPhysicalDMG ()  [inline]`

Gibt den physischen Schaden zurück.

Returns

Der physische Schaden.

Definition at line 31 of file Damage.cs.

```
00031                                   {
00032         return PhysicalDMG;
00033     }
```

4.5.3.2 GetPushAmount()

`Vector2 Damage.GetPushAmount ()  [inline]`

Gibt den Rückstoßvektor zurück.

Returns

Der Rückstoßvektor.

Definition at line 47 of file Damage.cs.

```
00047                                    {
00048         return PushAmount;
00049     }
```

4.5.3.3 GetSource()

`Node2D Damage.GetSource ()  [inline]`

Gibt die Ursache zurück.

Returns

Die Ursache.

Definition at line 55 of file Damage.cs.

```
00055                               {
00056         return Source;
00057     }
```

4.5.3.4 GetTrueDMG()

```
float Damage.GetTrueDMG ()  [inline]
```

Gibt den wahren Schaden zurück.

Returns

Der wahre Schaden.

Definition at line 39 of file Damage.cs.

```
00039                                {
00040         return TrueDMG;
00041     }
```

4.5.4 Member Data Documentation

4.5.4.1 PhysicalDMG

```
float Damage.PhysicalDMG  [private]
```

Definition at line 9 of file Damage.cs.

4.5.4.2 PushAmount

```
Vector2 Damage.PushAmount  [private]
```

Definition at line 11 of file Damage.cs.

4.5.4.3 Source

```
Node2D Damage.Source  [private]
```

Definition at line 12 of file Damage.cs.

4.5.4.4 TrueDMG

```
float Damage.TrueDMG  [private]
```

Definition at line 10 of file Damage.cs.

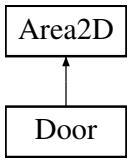
The documentation for this class was generated from the following file:

- Damage.cs

4.6 Door Class Reference

Klasse für die Tür.

Inheritance diagram for Door:



Public Member Functions

- override void _Ready ()
  Initialisierung der Node Spawn.

Public Attributes

- Node Spawn

Properties

- string DestinationLevelTag `[get, set]`
- string DestinationDoorTag `[get, set]`
- string SpawnDirection = "up" `[get, set]`

Private Member Functions

- void OnPlayerBodyEntered (Node body)
  Diese Funktion wird aufgerufen, wenn der Player die Tür betritt.

4.6.1 Detailed Description

Klasse für die Tür.

Die Klasse ist für den Wechsel zwischen den Levels zuständig.

Definition at line 8 of file Door.cs.

4.6.2 Member Function Documentation

4.6.2.1 _Ready()

```
override void Door._Ready ()  [inline]
```

Initialisierung der Node Spawn.

Definition at line 26 of file Door.cs.

```
00027     {
00028         Spawn = GetNode("Spawn");
00029     }
```

4.6.2.2 OnPlayerBodyEntered()

```
void Door.OnPlayerBodyEntered (
            Node body)  [inline], [private]
```

Diese Funktion wird aufgerufen, wenn der Player die Tür betritt.

Parameters

body	Der Körper, der die Tür betritt

Definition at line 36 of file Door.cs.

```
00037     {
00038         if (body is Player player)
00039         {
00040             var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00041             NavigationManager.GoToLevel(DestinationLevelTag, DestinationDoorTag);
00042         }
00043     }
```

4.6.3 Member Data Documentation

4.6.3.1 Spawn

```
Node Door.Spawn
```

Definition at line 10 of file Door.cs.

4.6.4 Property Documentation

4.6.4.1 DestinationDoorTag

```
string Door.DestinationDoorTag  [get], [set]
```

Definition at line 16 of file Door.cs.

```
00016 { get; set; }
```

4.6.4.2 DestinationLevelTag

```
string Door.DestinationLevelTag  [get], [set]
```

Definition at line 13 of file Door.cs.

```
00013 { get; set; }
```

4.6.4.3 SpawnDirection

```
string Door.SpawnDirection = "up"  [get], [set]
```

Definition at line 19 of file Door.cs.

```
00019 { get; set; } = "up";
```

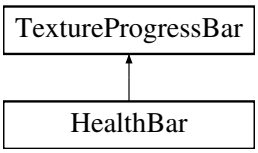
The documentation for this class was generated from the following file:

- Door.cs

4.7 HealthBar Class Reference

Klasse für die Gesundheitsleiste des Spielers. Synchronisiert die Anzeige der HealthBar mit den Lebenspunkten des Spielers.

Inheritance diagram for HealthBar:



Public Member Functions

- override void _Ready ()

  Initialisiert die HealthBar und verbindet sie mit den Lebenspunkten des Spielers. Lädt den Spieler-Knoten und setzt die maximale und aktuelle Gesundheit in der HealthBar.
- override void _Process (double DeltaTime)

  Aktualisiert die HealthBar in jedem Frame. Synchronisiert die Anzeige der aktuellen Lebenspunkte mit den Werten des Spielers.

4.7.1 Detailed Description

Klasse für die Gesundheitsleiste des Spielers. Synchronisiert die Anzeige der HealthBar mit den Lebenspunkten des Spielers.

Definition at line 7 of file HealthBar.cs.

4.7.2 Member Function Documentation

4.7.2.1 _Process()

```
override void HealthBar._Process (
            double DeltaTime)  [inline]
```

Aktualisiert die HealthBar in jedem Frame. Synchronisiert die Anzeige der aktuellen Lebenspunkte mit den Werten des Spielers.

Parameters

delta	Zeit seit dem letzten Frame (wird nicht direkt genutzt).

Definition at line 24 of file HealthBar.cs.

```
00024                                       {
00025         // Aktualisiere den Wert der HealthBar basierend auf der aktuellen Gesundheit des Spielers
00026         Value = PlayerStats.Instance.GetCurrentHealth();
00027     }
```

4.7.2.2 _Ready()

```
override void HealthBar._Ready ()  [inline]
```

Initialisiert die HealthBar und verbindet sie mit den Lebenspunkten des Spielers. Lädt den Spieler-Knoten und setzt die maximale und aktuelle Gesundheit in der HealthBar.

Definition at line 13 of file HealthBar.cs.

```
00013                               {
00014         // Setze die maximale Gesundheit der HealthBar basierend auf der Spieler-MaxHealth
00015         MaxValue = PlayerStats.Instance.GetMaxHealthPoints();
00016         Value = PlayerStats.Instance.GetCurrentHealth();
00017     }
```

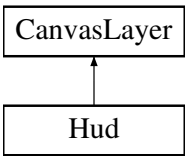
The documentation for this class was generated from the following file:

- HealthBar.cs

4.8 Hud Class Reference

Klasse für das PauseMenu.

Inheritance diagram for Hud:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- override void _Process (double DeltaTime)

  Methode wird in jedem Frame ausgeführt.
- void OnResumeButtonPressed ()

  Signal für den Resume-Button.
- void OnSaveButtonPressed ()

  Signal für den Save-Button.
- void OnSaveMenuButtonPressed ()

  Signal für den SaveAndReturnToMenu-Button.
- void OnSaveQuitButtonPressed ()

  Signal für den SaveAndQuit-Button.

Private Member Functions

- void TogglePause ()

  Toggled die Pause Funktion.

Private Attributes

- AnimationPlayer AnimationPlayer
- CenterContainer Buttons
- bool Enabled

4.8.1 Detailed Description

Klasse für das PauseMenu.

Definition at line 8 of file Hud.cs.

4.8.2 Member Function Documentation

4.8.2.1 _Process()

```
override void Hud._Process (
            double DeltaTime)  [inline]
```

Methode wird in jedem Frame ausgeführt.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 29 of file Hud.cs.

```
00029                                                {
00030         if(Input.IsActionJustPressed("escape")){
00031             TogglePause();
00032         }
00033     }
```

4.8.2.2 _Ready()

```
override void Hud._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 19 of file Hud.cs.

```
00019                                {
00020         AnimationPlayer = GetNode<AnimationPlayer>("PauseMenu/AnimationPlayer");
00021         Buttons = GetNode<CenterContainer>("PauseMenu/Buttons");
00022         AnimationPlayer.Play("RESET");
00023     }
```

4.8.2.3 OnResumeButtonPressed()

```
void Hud.OnResumeButtonPressed ()  [inline]
```

Signal für den Resume-Button.

Definition at line 53 of file Hud.cs.

```
00053                                        {
00054         TogglePause();
00055     }
```

4.8.2.4 OnSaveButtonPressed()

```
void Hud.OnSaveButtonPressed ()  [inline]
```

Signal für den Save-Button.

Definition at line 60 of file Hud.cs.

```
00060                                   {
00061         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00062     }
```

4.8.2.5 OnSaveMenuButtonPressed()

```
void Hud.OnSaveMenuButtonPressed ()  [inline]
```

Signal für den SaveAndReturnToMenu-Button.

Definition at line 67 of file Hud.cs.

```
00067                                           {
00068         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00069         NavigationManager.Instance.GoToLevel("main_menu", null);
00070         PlayerStats.Instance.Reload();
00071         GetTree().Paused = false;
00072     }
```

4.8.2.6 OnSaveQuitButtonPressed()

```
void Hud.OnSaveQuitButtonPressed ()  [inline]
```

Signal für den SaveAndQuit-Button.

Definition at line 77 of file Hud.cs.

```
00077                                           {
00078         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00079         GetTree().Quit();
00080     }
```

4.8.2.7 TogglePause()

```
void Hud.TogglePause ()  [inline], [private]
```

Toggled die Pause Funktion.

Definition at line 38 of file Hud.cs.

```
00038                                   {
00039         Enabled = !Enabled;
00040         GetTree().Paused = Enabled;
00041         if(Enabled){
00042             AnimationPlayer.Play("Pause");
00043             Buttons.Visible = true;
00044         } else {
00045             AnimationPlayer.PlayBackwards("Pause");
00046             Buttons.Visible = false;
00047         }
00048     }
```

4.8.3 Member Data Documentation

4.8.3.1 AnimationPlayer

```
AnimationPlayer Hud.AnimationPlayer  [private]
```

Definition at line 10 of file Hud.cs.

4.8.3.2 Buttons

```
CenterContainer Hud.Buttons  [private]
```

Definition at line 11 of file Hud.cs.

4.8.3.3 Enabled

```
bool Hud.Enabled  [private]
```

Definition at line 12 of file Hud.cs.

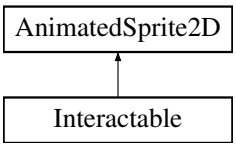
The documentation for this class was generated from the following file:

- Hud.cs

4.9 Interactable Class Reference

Klasse für Interaktion.

Inheritance diagram for Interactable:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- override void _Process (double DeltaTime)

  Testet, ob der Spieler mit der Node Interagiert und öffnet ein PopUp.
- void OnAreaBodyExited (Node2D Body)

  Detektiert, wenn der Spieler den Bereich verlässt und schließt das PopUp.

Properties

- String Text [get, set]

Private Attributes

- Player Player
- RichTextLabel TextLabel
- Control PopUp
- Area2D Area

4.9.1 Detailed Description

Klasse für Interaktion.

Definition at line 7 of file Interactable.cs.

4.9.2 Member Function Documentation

4.9.2.1 _Process()

```
override void Interactable._Process (
            double DeltaTime) [inline]
```

Testet, ob der Spieler mit der Node Interagiert und öffnet ein PopUp.

Parameters

DeltaTime	Zeit zwischen den Frames.

Definition at line 32 of file Interactable.cs.

```
00032                                                {
00033         if(Input.IsActionJustPressed("interact")){
00034             Godot.Collections.Array<Node2D> Bodies = Area.GetOverlappingBodies();
00035             foreach(Node2D Body in Bodies){
00036                 if(Body == Player){
00037                     TextLabel.Clear();
00038                     TextLabel.AppendText(Text);
00039                     PopUp.Visible = true;
00040                     return;
00041                 }
00042             }
00043         }
00044     }
```

4.9.2.2 _Ready()

```
override void Interactable._Ready () [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 21 of file Interactable.cs.

```
00021                                {
00022         Player = GetNode<Player>("../Player");
00023         TextLabel = GetNode<RichTextLabel>("../HUD/PopUp/Text");
00024         PopUp = GetNode<Control>("../HUD/PopUp");
00025         Area = GetNode<Area2D>("Area2D");
00026     }
```

4.9.2.3 OnAreaBodyExited()

```
void Interactable.OnAreaBodyExited (
            Node2D Body) [inline]
```

Detektiert, wenn der Spieler den Bereich verlässt und schließt das PopUp.

Parameters

Node2D	die den Bereich verlässt.

Definition at line 50 of file Interactable.cs.

```
00050                                                   {
00051         if(Body == Player){
00052             PopUp.Visible = false;
00053             TextLabel.Clear();
00054         }
00055     }
```

4.9.3 Member Data Documentation

4.9.3.1 Area

```
Area2D Interactable.Area  [private]
```

Definition at line 12 of file Interactable.cs.

4.9.3.2 Player

```
Player Interactable.Player  [private]
```

Definition at line 9 of file Interactable.cs.

4.9.3.3 PopUp

```
Control Interactable.PopUp  [private]
```

Definition at line 11 of file Interactable.cs.

4.9.3.4 TextLabel

```
RichTextLabel Interactable.TextLabel  [private]
```

Definition at line 10 of file Interactable.cs.

4.9.4 Property Documentation

4.9.4.1 Text

```
String Interactable.Text  [get], [set], [private]
```

Definition at line 15 of file Interactable.cs.

```
00015 { get; set;}
```

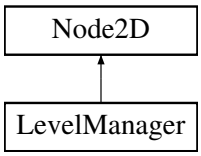
The documentation for this class was generated from the following file:

- Interactable.cs

4.10 LevelManager Class Reference

Klasse für den LevelManager Diese Klasse verwaltet den Levelwechsel und die Spielerpositionierung.

Inheritance diagram for LevelManager:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Private Member Functions

- void OnLevelSpawn (string DestinationTag)

  Wird aufgerufen, wenn ein neues Level geladen wird.

4.10.1 Detailed Description

Klasse für den LevelManager Diese Klasse verwaltet den Levelwechsel und die Spielerpositionierung.

Definition at line 7 of file LevelManager.cs.

4.10.2 Member Function Documentation

4.10.2.1 _Ready()

```
override void LevelManager._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Wenn ein Spawn-Tag gesetzt ist, wird der Spieler an die entsprechende Tür gesetzt. Dies wird verwendet, um den Spieler an eine bestimmte Tür zu setzen, wenn er von einem anderen Level aus spawnt.

Definition at line 13 of file LevelManager.cs.

```
00014     {
00015         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00016 
00021         if (NavigationManager.SpawnDoorTag != null)
00022         {
00023             OnLevelSpawn(NavigationManager.SpawnDoorTag);
00024         }
00025         else
00026         {
00027             NavigationManager.CallDeferred("TriggerPlayerSpawn", PlayerStats.Instance.GetPosition(),
      "");
00028         }
00029 
00030     }
```

4.10.2.2 OnLevelSpawn()

```
void LevelManager.OnLevelSpawn (
            string DestinationTag)  [inline], [private]
```

Wird aufgerufen, wenn ein neues Level geladen wird.

Parameters

DestinationTag	Das Tag der Tür, an der der Spieler spawnen soll.

Definition at line 36 of file LevelManager.cs.

```
00037     {
00038         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00039         // Pfad zur Tür basierend auf dem Ziel-Tag erstellen
00040         string DoorPath = "Doors/Door_" + DestinationTag;
00041 
00042         Door door = GetNode<Door>(DoorPath);
00043 
00044         // TriggerPlayerSpawn nach deferred ausführen
00045         NavigationManager.CallDeferred("TriggerPlayerSpawn", door.GlobalPosition,
      door.SpawnDirection);
00046     }
```

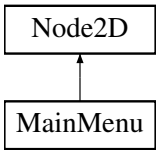
The documentation for this class was generated from the following file:

- LevelManager.cs

4.11 MainMenu Class Reference

Klasse für das MainMenu.

Inheritance diagram for MainMenu:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- void OnContinueButtonPressed ()

  Signal für den Continue-Button.
- void OnQuitButtonPressed ()

  Signal für den Quit-Button.
- void OnNewGameButtonPressed ()

  Signal für den NewGame-Button.
- void OnLoadGameButtonPressed ()

  Signal für den LoadGame-Button.
- void OnBackButtonPressed ()

  Signal für den Back-Button.
- void OnSave1SelectPressed ()

  Signal für den Select1-Button.
- void OnSave1DeletePressed ()

  Signal für den Delete1-Button.
- void OnSave2SelectPressed ()

  Signal für den Select2-Button.

- void OnSave2DeletePressed ()

  Signal für den Delete2-Button.
- void OnSave3SelectPressed ()

  Signal für den Select3-Button.
- void OnSave3DeletePressed ()

  Signal für den Delete3-Button.
- void OnDeleteConfirmationCanceled ()

  Signal für den Delete-Abbruch.
- void OnDeleteConfirmationConfirmed ()

  Signal für die Delete-Bestätigung.
- void OnDeleteConfirmationCloseRequested ()

  Signal für das Schließen der Delete-Bestätigung.

Private Member Functions

- void Change ()

  Wechselt das Menu zwischen den verschiedenen States.

Private Attributes

- int MenuState = 0
- VBoxContainer Navigation
- MarginContainer SavesContainer
- Button ContinueButton
- Label InfoLabel
- Label[ ] SaveLabel = new Label[3]
- Button[ ] SelectButton = new Button[3]
- Button[ ] DeleteButton = new Button[3]
- ConfirmationDialog DeleteConfirmation
- int SaveToDelete = 0

4.11.1 Detailed Description

Klasse für das MainMenu.

Definition at line 7 of file MainMenu.cs.

4.11.2 Member Function Documentation

4.11.2.1 _Ready()

```
override void MainMenu._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 25 of file MainMenu.cs.

```
00025                                {
00026         Navigation = GetNode<VBoxContainer>("Control/Navigation");
00027         SavesContainer = GetNode<MarginContainer>("Control/Saves");
00028         ContinueButton = GetNode<Button>("Control/Navigation/ContinueButton");
00029         InfoLabel = GetNode<Label>("Control/Saves/VBoxContainer/Info");
```

```
00030
00031         SaveLabel[0] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Label");
00032         SelectButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Select");
00033         DeleteButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Delete");
00034         SaveLabel[1] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Label");
00035         SelectButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Select");
00036         DeleteButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Delete");
00037         SaveLabel[2] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Label");
00038         SelectButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Select");
00039         DeleteButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Delete");
00040
00041         DeleteConfirmation = GetNode<ConfirmationDialog>("DeleteConfirmation");
00042
00043         if(StorageManager.Instance.GetLastSaveId() > -1){
00044             ContinueButton.Visible = true;
00045         }
00046     }
```

4.11.2.2 Change()

```
void MainMenu.Change ()  [inline], [private]
```

Wechselt das Menu zwischen den verschiedenen States.

Definition at line 52 of file MainMenu.cs.

```
00052                           {
00053         if(MenuState == 0){
00054             SavesContainer.Visible = false;
00055             Navigation.Visible = true;
00056         } else {
00057             Navigation.Visible = false;
00058             SavesContainer.Visible = true;
00059
00060             int Saves = StorageManager.Instance.GetSaves();
00061
00062             if(MenuState == 1){
00063                 InfoLabel.Text = "Select empty save to start a new Game";
00064                 for(int i = 0; i < 3; i++){
00065                     if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
00066                         SaveLabel[i].Text = "Save " + (i+1);
00067                         SelectButton[i].Disabled = true;
00068                         DeleteButton[i].Disabled = false;
00069                     } else {
00070                         SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
00071                         SelectButton[i].Disabled = false;
00072                         DeleteButton[i].Disabled = true;
00073                     }
00074                 }
00075             } else {
00076                 InfoLabel.Text = "Select save to load Game";
00077                 for(int i = 0; i < 3; i++){
00078                     if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
00079                         SaveLabel[i].Text = "Save " + (i+1);
00080                         SelectButton[i].Disabled = false;
00081                         DeleteButton[i].Disabled = false;
00082                     } else {
00083                         SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
00084                         SelectButton[i].Disabled = true;
00085                         DeleteButton[i].Disabled = true;
00086                     }
00087                 }
00088             }
00089         }
00090     }
```

4.11.2.3 OnBackButtonPressed()

```
void MainMenu.OnBackButtonPressed ()  [inline]
```

Signal für den Back-Button.

Definition at line 127 of file MainMenu.cs.

```
00127                                   {
00128         MenuState = 0;
00129         Change();
00130     }
```

4.11.2.4 OnContinueButtonPressed()

```
void MainMenu.OnContinueButtonPressed ()  [inline]
```

Signal für den Continue-Button.

Definition at line 95 of file MainMenu.cs.

```
00095                                       {
00096         StorageManager.Instance.LoadGameFile(StorageManager.Instance.GetLastSaveId());
00097         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00098     }
```

4.11.2.5 OnDeleteConfirmationCanceled()

```
void MainMenu.OnDeleteConfirmationCanceled ()  [inline]
```

Signal für den Delete-Abbruch.

Definition at line 198 of file MainMenu.cs.

```
00198                                                {
00199         SaveToDelete = 0;
00200         Change();
00201     }
```

4.11.2.6 OnDeleteConfirmationCloseRequested()

```
void MainMenu.OnDeleteConfirmationCloseRequested ()  [inline]
```

Signal für das Schließen der Delete-Bestätigung.

Definition at line 214 of file MainMenu.cs.

```
00214                                                    {
00215         OnDeleteConfirmationCanceled();
00216     }
```

4.11.2.7 OnDeleteConfirmationConfirmed()

```
void MainMenu.OnDeleteConfirmationConfirmed ()  [inline]
```

Signal für die Delete-Bestätigung.

Definition at line 206 of file MainMenu.cs.

```
00206                                                 {
00207         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() ^ (int) Math.Pow(2,
      SaveToDelete - 1));
00208         Change();
00209     }
```

4.11.2.8 OnLoadGameButtonPressed()

```
void MainMenu.OnLoadGameButtonPressed ()  [inline]
```

Signal für den LoadGame-Button.

Definition at line 119 of file MainMenu.cs.

```
00119                                       {
00120         MenuState = 2;
00121         Change();
00122     }
```

4.11.2.9 OnNewGameButtonPressed()

```
void MainMenu.OnNewGameButtonPressed ()  [inline]
```

Signal für den NewGame-Button.

Definition at line 111 of file MainMenu.cs.

```
00111                                       {
00112         MenuState = 1;
00113         Change();
00114     }
```

4.11.2.10 OnQuitButtonPressed()

```
void MainMenu.OnQuitButtonPressed ()  [inline]
```

Signal für den Quit-Button.

Definition at line 103 of file MainMenu.cs.

```
00103                                     {
00104         StorageManager.Instance.SaveSettings();
00105         GetTree().Quit();
00106     }
```

4.11.2.11 OnSave1DeletePressed()

```
void MainMenu.OnSave1DeletePressed ()  [inline]
```

Signal für den Delete1-Button.

Definition at line 147 of file MainMenu.cs.

```
00147                                       {
00148         SaveToDelete = 1;
00149         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00150         DeleteConfirmation.Show();
00151     }
```

4.11.2.12 OnSave1SelectPressed()

```
void MainMenu.OnSave1SelectPressed ()  [inline]
```

Signal für den Select1-Button.

Definition at line 135 of file MainMenu.cs.

```
00135                                       {
00136         if(MenuState == 2){
00137             StorageManager.Instance.LoadGameFile(0);
00138         }
00139         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00140         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 1);
00141         StorageManager.Instance.SetLastSaveId(0);
00142     }
```

4.11.2.13 OnSave2DeletePressed()

```
void MainMenu.OnSave2DeletePressed ()  [inline]
```

Signal für den Delete2-Button.

Definition at line 168 of file MainMenu.cs.

```
00168                                       {
00169         SaveToDelete = 2;
00170         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00171         DeleteConfirmation.Show();
00172     }
```

4.11.2.14 OnSave2SelectPressed()

```
void MainMenu.OnSave2SelectPressed ()  [inline]
```

Signal für den Select2-Button.

Definition at line 156 of file MainMenu.cs.

```
00156                                   {
00157         if(MenuState == 2){
00158             StorageManager.Instance.LoadGameFile(1);
00159         }
00160         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00161         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 2);
00162         StorageManager.Instance.SetLastSaveId(1);
00163     }
```

4.11.2.15 OnSave3DeletePressed()

```
void MainMenu.OnSave3DeletePressed ()  [inline]
```

Signal für den Delete3-Button.

Definition at line 189 of file MainMenu.cs.

```
00189                                   {
00190         SaveToDelete = 3;
00191         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00192         DeleteConfirmation.Show();
00193     }
```

4.11.2.16 OnSave3SelectPressed()

```
void MainMenu.OnSave3SelectPressed ()  [inline]
```

Signal für den Select3-Button.

Definition at line 177 of file MainMenu.cs.

```
00177                                   {
00178         if(MenuState == 2){
00179             StorageManager.Instance.LoadGameFile(2);
00180         }
00181         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00182         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 4);
00183         StorageManager.Instance.SetLastSaveId(2);
00184     }
```

4.11.3 Member Data Documentation

4.11.3.1 ContinueButton

```
Button MainMenu.ContinueButton  [private]
```

Definition at line 12 of file MainMenu.cs.

4.11.3.2 DeleteButton

```
Button [] MainMenu.DeleteButton = new Button[3]  [private]
```

Definition at line 16 of file MainMenu.cs.

4.11.3.3 DeleteConfirmation

```
ConfirmationDialog MainMenu.DeleteConfirmation  [private]
```

Definition at line 17 of file MainMenu.cs.

4.11.3.4 InfoLabel

```
Label MainMenu.InfoLabel  [private]
```

Definition at line 13 of file MainMenu.cs.

4.11.3.5 MenuState

```
int MainMenu.MenuState = 0  [private]
```

Definition at line 9 of file MainMenu.cs.

4.11.3.6 Navigation

```
VBoxContainer MainMenu.Navigation  [private]
```

Definition at line 10 of file MainMenu.cs.

4.11.3.7 SaveLabel

```
Label [] MainMenu.SaveLabel = new Label[3]  [private]
```

Definition at line 14 of file MainMenu.cs.

4.11.3.8 SavesContainer

```
MarginContainer MainMenu.SavesContainer  [private]
```

Definition at line 11 of file MainMenu.cs.

4.11.3.9 SaveToDelete

```
int MainMenu.SaveToDelete = 0  [private]
```

Definition at line 18 of file MainMenu.cs.

4.11.3.10 SelectButton

```
Button [] MainMenu.SelectButton = new Button[3]  [private]
```

Definition at line 15 of file MainMenu.cs.

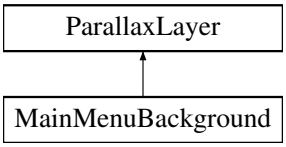
The documentation for this class was generated from the following file:

- MainMenu.cs

4.12 MainMenuBackground Class Reference

Klasse für die MainMenuBackground-Animation.

Inheritance diagram for MainMenuBackground:



Public Member Functions

- override void _Process (double DeltaTime)
  Methode wird in jedem Frame ausgeführt.

Private Attributes

- float ScrollSpeed = -10f

4.12.1 Detailed Description

Klasse für die MainMenuBackground-Animation.

Definition at line 7 of file MainMenuBackground.cs.

4.12.2 Member Function Documentation

4.12.2.1 _Process()

```
override void MainMenuBackground._Process (
            double DeltaTime)  [inline]
```

Methode wird in jedem Frame ausgeführt.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 16 of file MainMenuBackground.cs.

```
00016                                                   {
00017         float X = GetMotionOffset().X;
00018         X += ScrollSpeed * (float) DeltaTime;
00019         SetMotionOffset(new Vector2(X,0));
00020     }
```

4.12.3 Member Data Documentation

4.12.3.1 ScrollSpeed

```
float MainMenuBackground.ScrollSpeed = -10f  [private]
```

Definition at line 10 of file MainMenuBackground.cs.

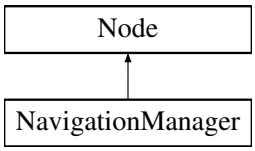
The documentation for this class was generated from the following file:

- MainMenuBackground.cs

4.13 NavigationManager Class Reference

Der NavigationManager ist für das Laden von Leveln und das Spawnen des Spielers verantwortlich. Der NavigationManager ist ein Singleton, der in der Haupt-Szene platziert wird und von anderen Skripten verwendet wird, um Level zu laden und den Spieler zu spawnen.

Inheritance diagram for NavigationManager:



Public Member Functions

- delegate void OnTriggerPlayerSpawnEventHandler (Vector2 Position, string Direction)

  Das Signal, das ausgelöst wird, wenn der Spieler spawnen soll.
- override void _Ready ()

  Initialisiert den NavigationManager und setzt ihn als Singleton.
- void GoToLevel (string LevelTag, string DestinationTag)

  Lädt das angegebene Level und setzt das Ziel-Tag für den Spieler-Spawn.
- void TriggerPlayerSpawn (Vector2 Position, string Direction)

  Lädt das angegebene Level und setzt das Ziel-Tag für den Spieler-Spawn.

Properties

- static NavigationManager Instance `[get, private set]`
- string SpawnDoorTag `[get, private set]`

Private Member Functions

- void DeferredChangeScene (PackedScene SceneToLoad)

  Diese Methode wird aufgerufen, um die Szene zu wechseln.

Static Private Attributes

- static readonly PackedScene SceneMainMenu = (PackedScene)GD.Load("res://Scenes/main_menu.tscn")
- static readonly PackedScene SceneIntro = (PackedScene)GD.Load("res://Scenes/intro.tscn")
- static readonly PackedScene SceneLevel1 = (PackedScene)GD.Load("res://Scenes/level1.tscn")
- static readonly PackedScene SceneBoss = (PackedScene)GD.Load("res://Scenes/bossRoom.tscn")
- static readonly PackedScene SceneLevelOne = (PackedScene)GD.Load("res://Scenes/level_one.tscn")
- static readonly PackedScene SceneLevelTwo = (PackedScene)GD.Load("res://Scenes/level_two.tscn")

4.13.1 Detailed Description

Der NavigationManager ist für das Laden von Leveln und das Spawnen des Spielers verantwortlich. Der NavigationManager ist ein Singleton, der in der Haupt-Szene platziert wird und von anderen Skripten verwendet wird, um Level zu laden und den Spieler zu spawnen.

Definition at line 7 of file NavigationManager.cs.

4.13.2 Member Function Documentation

4.13.2.1 _Ready()

```
override void NavigationManager._Ready ()  [inline]
```

Initialisiert den NavigationManager und setzt ihn als Singleton.

Definition at line 32 of file NavigationManager.cs.

```
00032                                   {
00033         Instance = this;
00034     }
```

4.13.2.2 DeferredChangeScene()

```
void NavigationManager.DeferredChangeScene (
            PackedScene SceneToLoad)  [inline], [private]
```

Diese Methode wird aufgerufen, um die Szene zu wechseln.

Parameters

SceneToLoad	Die Szene, die geladen werden soll.

Definition at line 83 of file NavigationManager.cs.

```
00084     {
00085         GetTree().ChangeSceneToPacked(SceneToLoad);
00086     }
```

4.13.2.3 GoToLevel()

```
void NavigationManager.GoToLevel (
            string LevelTag,
            string DestinationTag)  [inline]
```

Lädt das angegebene Level und setzt das Ziel-Tag für den Spieler-Spawn.

Parameters

LevelTag	Das Tag des Levels, das geladen werden soll.
DestinationTag	Das Tag der Tür, an der der Spieler spawnen soll.

Definition at line 41 of file NavigationManager.cs.

```
00042     {
00043         PackedScene SceneToLoad = null;
00044
00045         // Bestimmen, welches Level geladen werden soll
00046         switch (LevelTag)
00047         {
00048             case "main_menu":
00049                 SceneToLoad = SceneMainMenu;
00050                 break;
00051             case "intro":
00052                 SceneToLoad = SceneIntro;
00053                 break;
00054             case "level1":
00055                 SceneToLoad = SceneLevel1;
00056                 break;
00057             case "bossRoom":
00058                 SceneToLoad = SceneBoss;
00059                 break;
00060             case "level_one":
00061                 SceneToLoad = SceneLevelOne;
00062                 break;
00063             case "level_two":
00064                 SceneToLoad = SceneLevelTwo;
00065                 break;
00066         }
00067
00068         // Überprüfen, ob eine Szene ausgewählt wurde und diese dann laden
00069         if (SceneToLoad != null){
00070             if(SceneToLoad != SceneMainMenu){
00071                 PlayerStats.Instance.SetCurrentLevelTag(LevelTag);
00072                 SpawnDoorTag = DestinationTag;
00073             }
00074             // Verwendung der ChangeSceneToPacked-Methode in Godot 4
00075             CallDeferred(nameof(DeferredChangeScene), SceneToLoad);
00076         }
00077     }
```

4.13.2.4 OnTriggerPlayerSpawnEventHandler()

```
delegate void NavigationManager.OnTriggerPlayerSpawnEventHandler (
            Vector2 Position,
            string Direction)
```

Das Signal, das ausgelöst wird, wenn der Spieler spawnen soll.

Parameters

Position	Die Position, an der der Spieler spawnen soll.
Direction	Die Richtung, in die der Spieler schauen soll.

4.13.2.5 TriggerPlayerSpawn()

```
void NavigationManager.TriggerPlayerSpawn (
            Vector2 Position,
            string Direction)  [inline]
```

Lädt das angegebene Level und setzt das Ziel-Tag für den Spieler-Spawn.

Parameters

Position	Die Position, an der der Spieler spawnen soll.
Direction	Die Richtung, in die der Spieler schauen soll.

Definition at line 93 of file NavigationManager.cs.

```
00094     {
00095         EmitSignal(SignalName.OnTriggerPlayerSpawn, Position, Direction);
00096     }
```

4.13.3 Member Data Documentation

4.13.3.1 SceneBoss

```
readonly PackedScene NavigationManager.SceneBoss = (PackedScene)GD.Load("res://Scenes/boss↩
Room.tscn")  [static], [private]
```

Definition at line 14 of file NavigationManager.cs.

4.13.3.2 SceneIntro

```
readonly PackedScene NavigationManager.SceneIntro = (PackedScene)GD.Load("res://Scenes/intro.↩
tscn")  [static], [private]
```

Definition at line 12 of file NavigationManager.cs.

4.13.3.3 SceneLevel1

```
readonly PackedScene NavigationManager.SceneLevel1 = (PackedScene)GD.Load("res://Scenes/level1.↩
tscn")  [static], [private]
```

Definition at line 13 of file NavigationManager.cs.

4.13.3.4 SceneLevelOne

```
readonly PackedScene NavigationManager.SceneLevelOne = (PackedScene)GD.Load("res://Scenes/level↩
_one.tscn")  [static], [private]
```

Definition at line 15 of file NavigationManager.cs.

4.13.3.5 SceneLevelTwo

```
readonly PackedScene NavigationManager.SceneLevelTwo = (PackedScene)GD.Load("res://Scenes/level↩
_two.tscn")  [static], [private]
```

Definition at line 16 of file NavigationManager.cs.

4.13.3.6 SceneMainMenu

```
readonly PackedScene NavigationManager.SceneMainMenu = (PackedScene)GD.Load("res://Scenes/main↩
_menu.tscn")  [static], [private]
```

Definition at line 11 of file NavigationManager.cs.

4.13.4 Property Documentation

4.13.4.1 Instance

```
NavigationManager NavigationManager.Instance  [static], [get], [private set]
```

Definition at line 9 of file NavigationManager.cs.

```
00009 { get; private set; }
```

4.13.4.2 SpawnDoorTag

```
string NavigationManager.SpawnDoorTag  [get], [private set]
```

Definition at line 19 of file NavigationManager.cs.

```
00019 { get; private set; }
```

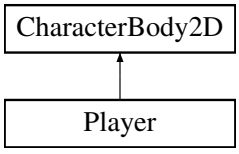
The documentation for this class was generated from the following file:

- NavigationManager.cs

4.14 Player Class Reference

Klasse für den Spielercharakter. Verwaltet Bewegung, Sprünge, Angriffe und Animationen.

Inheritance diagram for Player:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.
- override void _PhysicsProcess (double DeltaTime)

  Physikalische Prozesse werden in jedem Frame ausgeführt. Berechnet Gravitation, Bewegung, Sprünge und Dashes.
- bool IsAttacking ()

  Überprüft, ob der Spieler gerade angreift.
- bool IsBlocking ()

  Überprüft, ob der Spieler blockiert.
- void MaxHeal ()

  Heilt den Spieler vollständig, indem die aktuellen Lebenspunkte auf das Maximum gesetzt werden.
- void TakeDamage (Damage Damage)

  Wendet Schaden auf den Spieler an. Reduziert die aktuellen Lebenspunkte basierend auf dem übergebenen Schaden und wendet einen Rückstoßeffekt an.
- Damage GetDamage ()

  Gibt den Schaden zurück, den der Spieler mit seinem aktuellen Angriff verursacht. Der Schaden basiert auf der letzten Angriffsmethode (`light_attack` oder `heavy_attack`).
- void RegenerateStamina (float Amount, double delta)

  Regeneriert die Stamina des Spielers, wenn er für eine bestimmte Zeit keine Stamina-verbrauchende Aktion durchgeführt hat.
- bool UseStamina (float Amount)

  Verbraucht eine bestimmte Menge an Stamina, falls genügend verfügbar ist. Setzt den Inaktivitäts-Timer zurück, wenn Stamina verbraucht wird.
- void SlowPlayer (float SlowAmount)

  Verlangsamt den Spieler um einen bestimmten Prozentsatz.
- void Respawn ()

  Lässt den Spieler am Checkpoint spawnen.
- BloodVial GetBloodVials ()

  Getter für BloodVials.
- void SetSinAmount (int Value)

  Setzt den SinAmount des Spielers.

Private Member Functions

- void HandleJump ()

  Verarbeitet die Sprunglogik. Setzt den Sprungzähler zurück und ermöglicht einen Doppelsprung.
- void HandleMovement (double DeltaTime)

  Verarbeitet die Bewegung des Spielers. Regelt normale Bewegungen, Dashes und Kollisionen.
- void StartDash ()

  Startet den Dash-Prozess.
- void DashInProgress (double DeltaTime)

  Führt die Logik während eines Dashes aus.
- void CreateDashEffect ()

  Erstellt einen visuellen Dash-Trail. Der Spieler hinterlässt eine Spur während des Dashes.
- void StopDash ()

  Stoppt den Dash.
- void OnSpawn (Vector2 position, string direction)

  Wird aufgerufen, wenn der Spieler an einer neuen Position spawnen soll.
- void UpdateAnimations ()

  Aktualisiert die Animationen des Spielers.

Private Attributes

- int JumpMax = 2
- int JumpCount = 0
- Vector2 DashDirection = Vector2.Zero
- float DashSpeed = 300f
- bool IsDashing = false
- bool CanDash = true
- float DashTrailInterval = 0.05f
- float DashTrailTimer = 0f
- AnimationPlayer AnimationPlayer
- Sprite2D Sprite
- Timer DashEffect
- Timer DashTimer
- CollisionShape2D SwordCollision
- CollisionShape2D PlayerHitbox
- BloodVial BloodVials
- Label SinDisplay
- Vector2 HauptHitbox
- int LastAttack = 0
- float TimeSinceLastStaminaUse = 0f

Static Private Attributes

- const float SPEED = 100f
- const float JUMP_VELOCITY = -300f

4.14.1 Detailed Description

Klasse für den Spielercharakter. Verwaltet Bewegung, Sprünge, Angriffe und Animationen.

Definition at line 8 of file Player.cs.

4.14.2 Member Function Documentation

4.14.2.1 _PhysicsProcess()

```
override void Player._PhysicsProcess (
            double DeltaTime)  [inline]
```

Physikalische Prozesse werden in jedem Frame ausgeführt. Berechnet Gravitation, Bewegung, Sprünge und Dashes.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 67 of file Player.cs.

```
00067                                                                   {
00068         // Gravitation hinzufügen, wenn der Charakter nicht am Boden ist
00069         if (!IsOnFloor()) {
00070             Velocity += GetGravity() * (float)DeltaTime;
00071         } else {
00072             CanDash = true; // Dash wird zurückgesetzt, wenn der Charakter am Boden ist
00073         }
00074 
00075         TimeSinceLastStaminaUse += (float)DeltaTime;
00076         RegenerateStamina(20f, DeltaTime);
00077 
00078         // Heal
00079         if(Input.IsActionJustPressed("heal")){
00080             BloodVials.UseBloodVial();
00081         }
00082 
00083         HandleJump();
00084         HandleMovement(DeltaTime);
00085         MoveAndSlide();
00086         UpdateAnimations();
00087         PlayerStats.Instance.SetPosition(Position);
00088         }
```

4.14.2.2 _Ready()

```
override void Player._Ready ()  [inline]
```

Initialisierung der Referenzen. Findet die relevanten Knoten in der Szene und weist sie zu.

Definition at line 43 of file Player.cs.

```
00043                                 {
00044         AnimationPlayer = GetNode<AnimationPlayer>("AnimationPlayer");
00045         Sprite = GetNode<Sprite2D>("Sprite2D");
00046         DashEffect = GetNode<Timer>("DashEffect");
00047         DashTimer = GetNode<Timer>("DashTimer");
00048         SwordCollision = GetNode<CollisionShape2D>("Sprite2D/SwordHit/SwordCollision");
00049         PlayerHitbox = GetNode<CollisionShape2D>("PlayerHitbox");
00050         HauptHitbox = PlayerHitbox.Position;
00051         BloodVials = GetNode<BloodVial>("../HUD/BloodVial/Counter");
00052         SinDisplay = GetNode<Label>("../HUD/SinAmount/Counter");
00053 
00054         SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";
00055 
00056         NavigationManager navigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00057         navigationManager.Connect("OnTriggerPlayerSpawn", new Callable(this, nameof(OnSpawn)));
00058 
00059         Position = PlayerStats.Instance.GetPosition();
00060     }
```

4.14.2.3 CreateDashEffect()

```
void Player.CreateDashEffect ()  [inline], [private]
```

Erstellt einen visuellen Dash-Trail. Der Spieler hinterlässt eine Spur während des Dashes.

Definition at line 207 of file Player.cs.

```
00207                                       {
00208         Sprite2D PlayerCopyNode = (Sprite2D)Sprite.Duplicate();
00209         GetParent().AddChild(PlayerCopyNode);
00210 
00211         CollisionShape2D SwordCollisionCopy =
      PlayerCopyNode.GetNode<CollisionShape2D>("SwordHit/SwordCollision");
00212         if (SwordCollisionCopy != null) {
00213             SwordCollisionCopy.Disabled = true; // Deaktiviere die Kollision der Kopie
00214         }
00215 
00216         PlayerCopyNode.GlobalPosition = GlobalPosition + new Vector2(0, Sprite.Texture.GetHeight() *
      Sprite.Scale.Y * -0.5f);
```

```
00217
00218         // Verblassen-Effekt für den Dash-Trail hinzufügen
00219         float AnimationTime = (float)(DashTimer.WaitTime / 3);
00220
00221         Timer FadeTimer1 = new Timer();
00222         AddChild(FadeTimer1);
00223         FadeTimer1.Timeout += () => {
00224             if (IsInstanceValid(PlayerCopyNode)) {
00225                 PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.4f);
00226             }
00227         };
00228         FadeTimer1.Start(AnimationTime);
00229
00230         Timer FadeTimer2 = new Timer();
00231         AddChild(FadeTimer2);
00232         FadeTimer2.Timeout += () => {
00233             if (IsInstanceValid(PlayerCopyNode)) {
00234                 PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.2f);
00235             }
00236         };
00237         FadeTimer2.Start(AnimationTime * 2);
00238
00239         Timer FadeTimer3 = new Timer();
00240         AddChild(FadeTimer3);
00241         FadeTimer3.Timeout += () => {
00242             if (IsInstanceValid(PlayerCopyNode)) {
00243                 PlayerCopyNode.QueueFree();
00244             }
00245         };
00246         FadeTimer3.Start(AnimationTime * 3);
00247     }
```

4.14.2.4 DashInProgress()

```
void Player.DashInProgress (
            double DeltaTime)  [inline], [private]
```

Führt die Logik während eines Dashes aus.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 187 of file Player.cs.

```
00187                                                 {
00188         // Charakter bewegt sich in die Dash-Richtung mit Dash-Geschwindigkeit
00189         if (DashDirection == Vector2.Up) {
00190             Velocity = DashDirection / 1.5f * DashSpeed;
00191         } else {
00192             Velocity = DashDirection * DashSpeed;
00193         }
00194
00195         // Dash-Trail bei Intervallen erstellen
00196         DashTrailTimer -= (float)DeltaTime;
00197         if (DashTrailTimer <= 0f) {
00198             CreateDashEffect();
00199             DashTrailTimer = DashTrailInterval;
00200         }
00201     }
```

4.14.2.5 GetBloodVials()

```
BloodVial Player.GetBloodVials ()  [inline]
```

Getter für BloodVials.

Returns

BloodVial

Definition at line 383 of file Player.cs.

```
00383                                       {
00384         return BloodVials;
00385     }
```

4.14.2.6 GetDamage()

```
Damage Player.GetDamage ()  [inline]
```

Gibt den Schaden zurück, den der Spieler mit seinem aktuellen Angriff verursacht. Der Schaden basiert auf der letzten Angriffsmethode (`light_attack` oder `heavy_attack`).

Returns

Eine Instanz der Klasse `Damage`, die den physischen Schaden, wahren Schaden und Rückstoß enthält.

Definition at line 317 of file Player.cs.

```
00317                                   {
00318         if(LastAttack == 1){
00319             return new Damage(50, 0, Vector2.Zero, this);
00320         }
00321         if(LastAttack == 2){
00322             Vector2 Push = new Vector2(20,0);
00323             if(Sprite.FlipH){
00324                 Push = -Push;
00325             }
00326             return new Damage(100, 0, Push, this);
00327         }
00328         return new Damage(0,0,Vector2.Zero, this);
00329     }
```

4.14.2.7 HandleJump()

```
void Player.HandleJump ()  [inline], [private]
```

Verarbeitet die Sprunglogik. Setzt den Sprungzähler zurück und ermöglicht einen Doppelsprung.

Definition at line 94 of file Player.cs.

```
00094                                 {
00095         // Sprungzähler zurücksetzen, wenn der Charakter am Boden ist
00096         if (JumpCount != 0 && IsOnFloor()) {
00097             JumpCount = 0;
00098         }
00099
00100         // Überprüfen, ob der Sprung-Button gedrückt wurde und der Charakter noch Sprünge übrig hat
00101         if (Input.IsActionJustPressed("ui_up") && JumpCount < JumpMax) {
00102             if (JumpCount == 0) {
00103             // Erster Sprung ohne Stamina-Verlust
00104             Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
00105             JumpCount += 1;
00106             } else if (JumpCount > 0) {
00107                 // Beim Doppelsprung Stamina prüfen und abziehen
00108                 if (UseStamina(15)) {
00109                     Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
00110                     JumpCount += 1;
00111                 }
00112             }
00113         }
00114     }
```

4.14.2.8 HandleMovement()

```
void Player.HandleMovement (
             double DeltaTime)  [inline], [private]
```

Verarbeitet die Bewegung des Spielers. Regelt normale Bewegungen, Dashes und Kollisionen.

Parameters

DeltaTime	Zeit seit dem letzten Frame.

Definition at line 121 of file Player.cs.

```
00121                                                   {
00122         Vector2 direction = new Vector2(Input.GetAxis("ui_left", "ui_right"), Input.GetAxis("ui_up",
      "ui_down")).Normalized();
00123         float currentSpeed = SPEED;
00124 
00125         // Sprite umdrehen basierend auf der Bewegungsrichtung und Kollision umdrehen
00126         if (direction.X < 0) {
00127             Sprite.FlipH = true;
00128             SwordCollision.Position = new Vector2(-Mathf.Abs(SwordCollision.Position.X),
      SwordCollision.Position.Y);
00129             PlayerHitbox.Position = new Vector2(Sprite.Position.X * 1.8f, PlayerHitbox.Position.Y);
00130         } else if (direction.X > 0) {
00131             Sprite.FlipH = false;
00132             SwordCollision.Position = new Vector2(Mathf.Abs(SwordCollision.Position.X),
      SwordCollision.Position.Y);
00133             PlayerHitbox.Position = HauptHitbox;
00134         }
00135 
00136         // Geschwindigkeit reduzieren, wenn der Spieler angreift
00137         if (AnimationPlayer.CurrentAnimation == "light_attack") {
00138             currentSpeed *= 0.5f;
00139         } else if (AnimationPlayer.CurrentAnimation == "heavy_attack") {
00140             currentSpeed *= 0.15f;
00141         }
00142 
00143         // Blockieren stoppt die Bewegung
00144         if (IsBlocking()) {
00145             currentSpeed = 0;
00146         }
00147 
00148         if (IsDashing) {
00149             DashInProgress(DeltaTime);
00150         } else {
00151             // Normale Bewegung verarbeiten, wenn kein Dash aktiv ist
00152             if (direction != Vector2.Zero) {
00153                 Velocity = new Vector2(direction.X * currentSpeed, Velocity.Y);
00154             } else {
00155                 Velocity = new Vector2(Mathf.MoveToward(Velocity.X, 0, SPEED), Velocity.Y);
00156             }
00157 
00158             // Überprüfen, ob der Dash-Button gedrückt wurde mit eine Bewegungsrichtung und nicht
      schon am angreifen ist
00159             if (Input.IsActionJustPressed("dash") && direction != Vector2.Zero && CanDash &&
      !IsAttacking()) {
00160                 // Wenn der Player genug Stamina hat kann er dashen
00161                 if (UseStamina(20)){
00162                     DashDirection = direction;
00163                     StartDash();
00164                 }
00165             }
00166         }
00167     }
```

4.14.2.9 IsAttacking()

```
bool Player.IsAttacking ()  [inline]
```

Überprüft, ob der Spieler gerade angreift.

Returns

true, wenn der Spieler angreift.

Definition at line 265 of file Player.cs.

```
00265                                {
00266         return AnimationPlayer.CurrentAnimation == "heavy_attack" || AnimationPlayer.CurrentAnimation
      == "light_attack";
00267     }
```

4.14.2.10 IsBlocking()

```
bool Player.IsBlocking ()  [inline]
```

Überprüft, ob der Spieler blockiert.

Returns

true, wenn der Spieler blockiert.

Definition at line 273 of file Player.cs.

```
00273                           {
00274         return AnimationPlayer.CurrentAnimation == "block";
00275     }
```

4.14.2.11 MaxHeal()

```
void Player.MaxHeal ()  [inline]
```

Heilt den Spieler vollständig, indem die aktuellen Lebenspunkte auf das Maximum gesetzt werden.

Definition at line 280 of file Player.cs.

```
00280                       {
00281         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetMaxHealthPoints());
00282     }
```

4.14.2.12 OnSpawn()

```
void Player.OnSpawn (
            Vector2 position,
            string direction)  [inline], [private]
```

Wird aufgerufen, wenn der Spieler an einer neuen Position spawnen soll.

Parameters

position	Die Position, an der der Spieler spawnen soll.
direction	Die Richtung, in die der Spieler schauen soll.

Definition at line 402 of file Player.cs.

```
00402                                                       {
00403
00404         // Spielerposition auf die übergebene Position setzen
00405         if (direction == "right")
00406         {
00407             // Update the x value by adding 50 to it, keep the original y value
00408             Sprite.FlipH = false;
00409             position = position with { X = position.X + 25 };
00410         }
00411         else if (direction == "left")
00412         {
00413             // Update the x value by subtracting 50 from it, keep the original y value
00414             Sprite.FlipH = true;
00415             position = position with { X = position.X - 25 };
00416         }
00417         Position = position;
00418
00419     }
```

4.14.2.13 RegenerateStamina()

```
void Player.RegenerateStamina (
            float Amount,
            double delta)  [inline]
```

Regeneriert die Stamina des Spielers, wenn er für eine bestimmte Zeit keine Stamina-verbrauchende Aktion durchgeführt hat.

Parameters

Amount	Menge der Stamina, die regeneriert werden soll.
delta	Zeit seit dem letzten Frame.

Definition at line 336 of file Player.cs.

```
00336                                                           {
00337         // Wenn die Verzögerungszeit erreicht wurde, regeneriere Stamina
00338         if (TimeSinceLastStaminaUse >= 1f) {
00339             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() + Amount *
      (float)delta); // Regeneriere Stamina abhängig von der Zeit
00340         }
00341     }
```

4.14.2.14 Respawn()

```
void Player.Respawn ()  [inline]
```

Lässt den Spieler am Checkpoint spawnen.

Definition at line 372 of file Player.cs.

```
00372                         {
00373         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00374         NavigationManager.GoToLevel(PlayerStats.Instance.GetRespawnLevelTag(), "spawn");
00375         BloodVials.ResetUses();
00376
00377     }
```

4.14.2.15 SetSinAmount()

```
void Player.SetSinAmount (
             int Value)  [inline]
```

Setzt den SinAmount des Spielers.

Parameters

Value	Der neue Wert für den SinAmount.

Definition at line 391 of file Player.cs.

```
00391                                       {
00392         // SinAmount muss immer >= 0 sein
00393         PlayerStats.Instance.SetSinAmount(Value);
00394         SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";
00395     }
```

4.14.2.16 SlowPlayer()

```
void Player.SlowPlayer (
             float SlowAmount)  [inline]
```

Verlangsamt den Spieler um einen bestimmten Prozentsatz.

Parameters

SlowAmount	Der Prozentsatz, um den der Spieler verlangsamt werden soll.

Definition at line 365 of file Player.cs.

```
00365                                           {
00366         Velocity = new Vector2(Velocity.X * SlowAmount, Velocity.Y);
00367     }
```

4.14.2.17 StartDash()

```
void Player.StartDash ()  [inline], [private]
```

Startet den Dash-Prozess.

Definition at line 172 of file Player.cs.

```
00172                               {
00173         SetCollisionLayerValue(1,false);
00174         SetCollisionMaskValue(1,false);
00175         IsDashing = true;
00176         CanDash = false;
00177         DashTimer.Timeout += StopDash;
00178         DashTimer.Start();
00179         DashEffect.Start();
00180         DashTrailTimer = 0f;
00181     }
```

4.14.2.18 StopDash()

```
void Player.StopDash ()  [inline], [private]
```

Stoppt den Dash.

Definition at line 252 of file Player.cs.

```
00252                              {
00253         IsDashing = false;
00254         DashEffect.Stop();
00255         DashTimer.Stop();
00256         DashTimer.Timeout -= StopDash;
00257         SetCollisionLayerValue(1,true);
00258         SetCollisionMaskValue(1,true);
00259     }
```

4.14.2.19 TakeDamage()

```
void Player.TakeDamage (
            Damage Damage)  [inline]
```

Wendet Schaden auf den Spieler an. Reduziert die aktuellen Lebenspunkte basierend auf dem übergebenen Schaden und wendet einen Rückstoßeffekt an.

Parameters

Damage	Instanz der Klasse `Damage`, die den physischen und wahren Schaden sowie den Rückstoß enthält.

Definition at line 289 of file Player.cs.

```
00289                                       {
00290         float totalDamage = Damage.GetTrueDMG();
00291         if(!IsBlocking()){
00292             totalDamage += Damage.GetPhysicalDMG();
00293         } else {
00294             float CurrentStamina = PlayerStats.Instance.GetStamina();
00295             CurrentStamina -= Damage.GetPhysicalDMG();
00296             if(CurrentStamina < 0){
00297                 totalDamage -= CurrentStamina;
00298             }
00299             PlayerStats.Instance.SetStamina(CurrentStamina);
00300         }
00301
00302         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() - totalDamage);
00303         Position += Damage.GetPushAmount();
00304
00305         // Überprüfe, ob der Spieler gestorben ist
00306         if (PlayerStats.Instance.GetCurrentHealth() <= 0){
00307             GD.Print("Spieler ist gestorben!");
00308             Respawn();
00309         }
00310     }
```

4.14.2.20 UpdateAnimations()

```
void Player.UpdateAnimations ()  [inline], [private]
```

Aktualisiert die Animationen des Spielers.

Definition at line 425 of file Player.cs.

```
00425                                   {
00426         if (Input.IsActionJustPressed("light_attack") && !IsDashing && !IsAttacking()) {
00427             if (UseStamina(10)){
00428                 LastAttack = 1;
00429                 AnimationPlayer.Play("light_attack");
00430             }
00431         } else if (Input.IsActionJustPressed("heavy_attack") && !IsDashing && !IsAttacking()) {
00432             if (UseStamina(25)){
00433                 LastAttack = 2;
00434                 AnimationPlayer.Play("heavy_attack");
00435             }
00436         }
00437         if (Input.IsActionPressed("block") && !IsDashing && !IsAttacking() && IsOnFloor()) {
00438             if (UseStamina(0)){
00439                 AnimationPlayer.Play("block");
00440                 LastAttack = 0;
00441             }
00442         }
00443 
00444         if (IsOnFloor() && !IsAttacking() && !IsBlocking()) {
00445             LastAttack = 0;
00446             if (Velocity.X == 0) {
00447                 AnimationPlayer.Play("idle");
00448             } else {
00449                 AnimationPlayer.Play("run");
00450             }
00451         } else if (!IsOnFloor() && !IsAttacking() && !IsBlocking()) {
00452             LastAttack = 0;
00453             if (Velocity.Y < 0) {
00454                 AnimationPlayer.Play("jump");
00455             } else if (Velocity.Y > 0) {
00456                 AnimationPlayer.Play("fall");
00457             }
00458         }
00459     }
```

4.14.2.21 UseStamina()

```
bool Player.UseStamina (
             float Amount)  [inline]
```

Verbraucht eine bestimmte Menge an Stamina, falls genügend verfügbar ist. Setzt den Inaktivitäts-Timer zurück, wenn Stamina verbraucht wird.

Parameters

Amount	Die Menge an Stamina, die verbraucht werden soll.

Returns

`true`, wenn genügend Stamina verfügbar war und die Aktion ausgeführt wurde; andernfalls `false`.

Definition at line 349 of file Player.cs.

```
00349                                       {
00350         // Versucht, eine bestimmte Menge an Stamina zu verbrauchen.
00351         // Gibt true zurück, wenn genug Stamina verfügbar war; andernfalls false.
00352         if (PlayerStats.Instance.GetStamina() >= Amount) {
00353             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() - Amount);
00354             TimeSinceLastStaminaUse = 0f;
00355             return true;
00356         }
00357 
00358         return false;
00359     }
```

4.14.3 Member Data Documentation

4.14.3.1 AnimationPlayer

```
AnimationPlayer Player.AnimationPlayer  [private]
```

Definition at line 24 of file Player.cs.

4.14.3.2 BloodVials

```
BloodVial Player.BloodVials  [private]
```

Definition at line 30 of file Player.cs.

4.14.3.3 CanDash

```
bool Player.CanDash = true  [private]
```

Definition at line 19 of file Player.cs.

4.14.3.4 DashDirection

```
Vector2 Player.DashDirection = Vector2.Zero  [private]
```

Definition at line 16 of file Player.cs.

4.14.3.5 DashEffect

```
Timer Player.DashEffect  [private]
```

Definition at line 26 of file Player.cs.

4.14.3.6 DashSpeed

```
float Player.DashSpeed = 300f  [private]
```

Definition at line 17 of file Player.cs.

4.14.3.7 DashTimer

```
Timer Player.DashTimer  [private]
```

Definition at line 27 of file Player.cs.

4.14.3.8 DashTrailInterval

```
float Player.DashTrailInterval = 0.05f  [private]
```

Definition at line 20 of file Player.cs.

4.14.3.9 DashTrailTimer

```
float Player.DashTrailTimer = 0f  [private]
```

Definition at line 21 of file Player.cs.

4.14.3.10 HauptHitbox

```
Vector2 Player.HauptHitbox  [private]
```

Definition at line 33 of file Player.cs.

4.14.3.11 IsDashing

```
bool Player.IsDashing = false  [private]
```

Definition at line 18 of file Player.cs.

4.14.3.12 JUMP_VELOCITY

```
const float Player.JUMP_VELOCITY = -300f  [static], [private]
```

Definition at line 12 of file Player.cs.

4.14.3.13 JumpCount

```
int Player.JumpCount = 0  [private]
```

Definition at line 14 of file Player.cs.

4.14.3.14 JumpMax

```
int Player.JumpMax = 2  [private]
```

Definition at line 13 of file Player.cs.

4.14.3.15 LastAttack

```
int Player.LastAttack = 0  [private]
```

Definition at line 34 of file Player.cs.

4.14.3.16 PlayerHitbox

```
CollisionShape2D Player.PlayerHitbox  [private]
```

Definition at line 29 of file Player.cs.

4.14.3.17 SinDisplay

```
Label Player.SinDisplay  [private]
```

Definition at line 31 of file Player.cs.

4.14.3.18 SPEED

```
const float Player.SPEED = 100f  [static], [private]
```

Definition at line 11 of file Player.cs.

4.14.3.19 Sprite

```
Sprite2D Player.Sprite  [private]
```

Definition at line 25 of file Player.cs.

4.14.3.20 SwordCollision

```
CollisionShape2D Player.SwordCollision  [private]
```

Definition at line 28 of file Player.cs.

4.14.3.21 TimeSinceLastStaminaUse

```
float Player.TimeSinceLastStaminaUse = 0f  [private]
```

Definition at line 37 of file Player.cs.

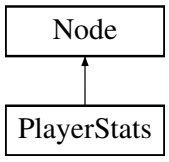
The documentation for this class was generated from the following file:

- Player.cs

4.15 PlayerStats Class Reference

Klasse für die Spielerstats.

Inheritance diagram for PlayerStats:



Public Member Functions

- override void _Ready ()

  Initialisierung der Referenzen.
- String GetRespawnLevelTag ()

  Getter für RespawnLevelTag.
- void SetRespawnLevelTag (String levelTag)

  Setter für RespawnLevelTag.
- String GetCurrentLevelTag ()

  Getter für CurrentLevelTag.
- void SetCurrentLevelTag (String levelTag)

  Setter für CurrentLevelTag.
- void SetSpawnPoint (Vector2 spawnPoint)

  Setzt den SpawnPoint des Spielers.
- Vector2 GetSpawnPoint ()

  Getter für den SpawnPoint.
- void SetPosition (Vector2 position)

  Setzt die Position des Spielers.
- Vector2 GetPosition ()

  Getter für die Position.
- int GetSinAmount ()

  Getter für SinAmount.
- void SetSinAmount (int Value)

  Setzt den SinAmount des Spielers.
- float GetMaxHealthPoints ()

  Gibt die maximalen Lebenspunkte des Spielers zurück.
- void SetMaxHealthPoints (float maxHealthPoints)

  Setzt die maximalen Lebenspunkte des Spielers.
- float GetCurrentHealth ()

  Gibt die aktuellen Lebenspunkte des Spielers zurück.
- void SetCurrentHealth (float Health)

  Setzt die aktuellen Lebenspunkte des Spielers.
- void SetMaxStamina (float Value)

  Setzt die maximale Stamina des Spielers.
- float GetMaxStamina ()

  Gibt die maximale Stamina des Spielers zurück.
- void SetStamina (float Value)

  Setzt die Stamina des Spielers.
- float GetStamina ()

  Gibt die aktuelle Stamina des Spielers zurück.
- void SetBVHealAmount (int Value)

  Setzt den HealAmount eines Bloodvials.
- int GetBVHealAmount ()

  Gibt den aktuellen HealAmount eines Bloodvials zurück.
- void SetBVMaxUses (int Value)

  Setzt die MaxUses der Bloodvials.
- int GetBVMaxUses ()

  Gibt die MaxUses der Bloodvials zurück.
- void SetBVCurrentUses (int Value)

  Setzt die CurrentUses der Bloodvials.
- int GetBVCurrentUses ()

  Gibt die CurrentUses der Bloodvials zurück.
- void Reload ()

  Setzt die Attribute zurück.

Properties

- static PlayerStats Instance `[get, private set]`

Private Attributes

- String RespawnLevelTag = "intro"
- String CurrentLevelTag = "intro"
- Vector2 SpawnPoint
- Vector2 Position = new Vector2(-540, 160)
- int SinAmount
- float MaxHealthPoints = 100f
- float CurrentHealth
- float MaxStamina = 100f
- float CurrentStamina
- int BVHealAmount = 25
- int BVMaxUses = 5
- int BVCurrentUses

4.15.1 Detailed Description

Klasse für die Spielerstats.

Definition at line 7 of file PlayerStats.cs.

4.15.2 Member Function Documentation

4.15.2.1 _Ready()

```
override void PlayerStats._Ready ()  [inline]
```

Initialisierung der Referenzen.

Definition at line 29 of file PlayerStats.cs.

```
00029                           {
00030         CurrentHealth = MaxHealthPoints;
00031         CurrentStamina = MaxStamina;
00032         BVCurrentUses = BVMaxUses;
00033         Instance = this;
00034     }
```

4.15.2.2 GetBVCurrentUses()

```
int PlayerStats.GetBVCurrentUses ()  [inline]
```

Gibt die CurrentUses der Bloodvials zurück.

Returns

Die aktuellen CurrentUses.

Definition at line 230 of file PlayerStats.cs.

```
00230                                {
00231         return BVCurrentUses;
00232     }
```

4.15.2.3 GetBVHealAmount()

```
int PlayerStats.GetBVHealAmount ()  [inline]
```

Gibt den aktuellen HealAmount eines Bloodvials zurück.

Returns

Der aktuelle HealAmount.

Definition at line 198 of file PlayerStats.cs.

```
00198                                   {
00199         return BVHealAmount;
00200     }
```

4.15.2.4 GetBVMaxUses()

```
int PlayerStats.GetBVMaxUses ()  [inline]
```

Gibt die MaxUses der Bloodvials zurück.

Returns

Die aktuellen MaxUses.

Definition at line 214 of file PlayerStats.cs.

```
00214                                   {
00215         return BVMaxUses;
00216     }
```

4.15.2.5 GetCurrentHealth()

```
float PlayerStats.GetCurrentHealth ()  [inline]
```

Gibt die aktuellen Lebenspunkte des Spielers zurück.

Returns

Die aktuellen Lebenspunkte.

Definition at line 139 of file PlayerStats.cs.

```
00139                                   {
00140         return CurrentHealth;
00141     }
```

4.15.2.6 GetCurrentLevelTag()

```
String PlayerStats.GetCurrentLevelTag ()  [inline]
```

Getter für CurrentLevelTag.

Returns

String CurrentLevelTag

Definition at line 56 of file PlayerStats.cs.

```
00056                                   {
00057         return CurrentLevelTag;
00058     }
```

4.15.2.7 GetMaxHealthPoints()

```
float PlayerStats.GetMaxHealthPoints ()  [inline]
```

Gibt die maximalen Lebenspunkte des Spielers zurück.

Returns

Die maximalen Lebenspunkte.

Definition at line 122 of file PlayerStats.cs.

```
00122                                   {
00123         return MaxHealthPoints;
00124     }
```

4.15.2.8 GetMaxStamina()

```
float PlayerStats.GetMaxStamina ()  [inline]
```

Gibt die maximale Stamina des Spielers zurück.

Returns

Die maximale Stamina.

Definition at line 165 of file PlayerStats.cs.

```
00165                                  {
00166         return MaxStamina;
00167     }
```

4.15.2.9 GetPosition()

```
Vector2 PlayerStats.GetPosition ()  [inline]
```

Getter für die Position.

Returns

Position des Spielers

Definition at line 96 of file PlayerStats.cs.

```
00096                                 {
00097         return Position;
00098     }
```

4.15.2.10 GetRespawnLevelTag()

```
String PlayerStats.GetRespawnLevelTag ()  [inline]
```

Getter für RespawnLevelTag.

Returns

String RespawnLevelTag

Definition at line 40 of file PlayerStats.cs.

```
00040                                      {
00041         return RespawnLevelTag;
00042     }
```

4.15.2.11 GetSinAmount()

```
int PlayerStats.GetSinAmount ()  [inline]
```

Getter für SinAmount.

Returns

int Sins

Definition at line 105 of file PlayerStats.cs.

```
00105                                {
00106         return SinAmount;
00107     }
```

4.15.2.12 GetSpawnPoint()

```
Vector2 PlayerStats.GetSpawnPoint ()  [inline]
```

Getter für den SpawnPoint.

Returns

Der SpawnPoint des Spielers

Definition at line 80 of file PlayerStats.cs.

```
00080                                    {
00081         return SpawnPoint;
00082     }
```

4.15.2.13 GetStamina()

```
float PlayerStats.GetStamina ()  [inline]
```

Gibt die aktuelle Stamina des Spielers zurück.

Returns

Die aktuelle Stamina.

Definition at line 182 of file PlayerStats.cs.

```
00182                                {
00183         return CurrentStamina;
00184     }
```

4.15.2.14 Reload()

```
void PlayerStats.Reload ()  [inline]
```

Setzt die Attribute zurück.

Definition at line 237 of file PlayerStats.cs.

```
00237                         {
00238         Instance = new PlayerStats();
00239         Instance._Ready();
00240     }
```

4.15.2.15 SetBVCurrentUses()

```
void PlayerStats.SetBVCurrentUses (
             int Value)  [inline]
```

Setzt die CurrentUses der Bloodvials.

Parameters

Value	Die CurrentUses der Bloodvials.

Definition at line 222 of file PlayerStats.cs.

```
00222                                           {
00223         BVCurrentUses = Math.Max(0, Value);
00224     }
```

4.15.2.16 SetBVHealAmount()

```
void PlayerStats.SetBVHealAmount (
             int Value)  [inline]
```

Setzt den HealAmount eines Bloodvials.

Parameters

Value	Den neuen Wert für den HealAmount.

Definition at line 190 of file PlayerStats.cs.

```
00190                                           {
00191         BVHealAmount = Math.Max(0, Value);
00192     }
```

4.15.2.17 SetBVMaxUses()

```
void PlayerStats.SetBVMaxUses (
             int Value)  [inline]
```

Setzt die MaxUses der Bloodvials.

Parameters

Value	Die MaxUses der Bloodvials.

Definition at line 206 of file PlayerStats.cs.

```
00206                                           {
00207         BVMaxUses = Math.Max(0, Value);
00208     }
```

4.15.2.18 SetCurrentHealth()

```
void PlayerStats.SetCurrentHealth (
             float Health)  [inline]
```

Setzt die aktuellen Lebenspunkte des Spielers.

Parameters

Health	Neue Lebenspunkte, die gesetzt werden sollen.

Definition at line 147 of file PlayerStats.cs.

```
00147                                           {
00148         // CurrentHealth darf MaxHealthPoints nicht überschreiten.
00149         CurrentHealth = Mathf.Min(Health, MaxHealthPoints);
00150     }
```

4.15.2.19 SetCurrentLevelTag()

```
void PlayerStats.SetCurrentLevelTag (
            String levelTag)  [inline]
```

Setter für CurrentLevelTag.

Parameters

CurrentLevelTag	Neuer Tag

Definition at line 64 of file PlayerStats.cs.

```
00064                                                {
00065         CurrentLevelTag = levelTag;
00066     }
```

4.15.2.20 SetMaxHealthPoints()

```
void PlayerStats.SetMaxHealthPoints (
            float maxHealthPoints)  [inline]
```

Setzt die maximalen Lebenspunkte des Spielers.

Parameters

maxHealthPoints	Die neuen maximalen Lebenspunkte (muss positiv sein).

Definition at line 130 of file PlayerStats.cs.

```
00130                                                    {
00131         // MaxHealthPoints muss immer positiv sein
00132         MaxHealthPoints = Mathf.Max(maxHealthPoints, 1); // Verhindert, dass MaxHealthPoints <= 0 wird
00133     }
```

4.15.2.21 SetMaxStamina()

```
void PlayerStats.SetMaxStamina (
            float Value)  [inline]
```

Setzt die maximale Stamina des Spielers.

Parameters

Value	Den neuen Wert für die maximale Stamina (muss positiv sein).

Definition at line 156 of file PlayerStats.cs.

```
00156                                        {
00157         // MaxStamina muss immer positiv sein
00158         MaxStamina = Mathf.Max(Value, 1);
00159     }
```

4.15.2.22 SetPosition()

```
void PlayerStats.SetPosition (
            Vector2 position)  [inline]
```

Setzt die Position des Spielers.

Parameters

Position	des Spielers.

Definition at line 88 of file PlayerStats.cs.

```
00088                                                   {
00089         Position = position;
00090     }
```

4.15.2.23 SetRespawnLevelTag()

```
void PlayerStats.SetRespawnLevelTag (
             String levelTag)  [inline]
```

Setter für RespawnLevelTag.

Parameters

RespawnLevelTag	Neuer Tag

Definition at line 48 of file PlayerStats.cs.

```
00048                                                   {
00049         RespawnLevelTag = levelTag;
00050     }
```

4.15.2.24 SetSinAmount()

```
void PlayerStats.SetSinAmount (
             int Value)  [inline]
```

Setzt den SinAmount des Spielers.

Parameters

Value	Der neue Wert für den SinAmount.

Definition at line 113 of file PlayerStats.cs.

```
00113                                                   {
00114         // SinAmount muss immer >= 0 sein
00115         SinAmount = Mathf.Max(Value, 0);
00116     }
```

4.15.2.25 SetSpawnPoint()

```
void PlayerStats.SetSpawnPoint (
             Vector2 spawnPoint)  [inline]
```

Setzt den SpawnPoint des Spielers.

Parameters

Der	SpawnPoint des Spielers.

Definition at line 72 of file PlayerStats.cs.

```
00072                                                   {
00073         SpawnPoint = spawnPoint;
00074     }
```

4.15.2.26 SetStamina()

```
void PlayerStats.SetStamina (
            float Value)  [inline]
```

Setzt die Stamina des Spielers.

Parameters

Value	Den neuen Wert für Stamina (muss im Bereich zwischen 0 und MaxStamina liegen).

Definition at line 173 of file PlayerStats.cs.

```
00173                                       {
00174         // Stellt sicher, dass die CurrentStamina im gültigen Bereich bleibt (zwischen 0 und
      MaxStamina)
00175         CurrentStamina = Mathf.Clamp(Value, 0, MaxStamina);
00176     }
```

4.15.3 Member Data Documentation

4.15.3.1 BVCurrentUses

```
int PlayerStats.BVCurrentUses  [private]
```

Definition at line 23 of file PlayerStats.cs.

4.15.3.2 BVHealAmount

```
int PlayerStats.BVHealAmount = 25  [private]
```

Definition at line 21 of file PlayerStats.cs.

4.15.3.3 BVMaxUses

```
int PlayerStats.BVMaxUses = 5  [private]
```

Definition at line 22 of file PlayerStats.cs.

4.15.3.4 CurrentHealth

```
float PlayerStats.CurrentHealth  [private]
```

Definition at line 18 of file PlayerStats.cs.

4.15.3.5 CurrentLevelTag

```
String PlayerStats.CurrentLevelTag = "intro"  [private]
```

Definition at line 13 of file PlayerStats.cs.

4.15.3.6 CurrentStamina

```
float PlayerStats.CurrentStamina  [private]
```

Definition at line 20 of file PlayerStats.cs.

4.15.3.7 MaxHealthPoints

```
float PlayerStats.MaxHealthPoints = 100f  [private]
```

Definition at line 17 of file PlayerStats.cs.

4.15.3.8 MaxStamina

```
float PlayerStats.MaxStamina = 100f  [private]
```

Definition at line 19 of file PlayerStats.cs.

4.15.3.9 Position

```
Vector2 PlayerStats.Position = new Vector2(-540, 160)  [private]
```

Definition at line 15 of file PlayerStats.cs.

4.15.3.10 RespawnLevelTag

```
String PlayerStats.RespawnLevelTag = "intro"  [private]
```

Definition at line 12 of file PlayerStats.cs.

4.15.3.11 SinAmount

```
int PlayerStats.SinAmount  [private]
```

Definition at line 16 of file PlayerStats.cs.

4.15.3.12 SpawnPoint

```
Vector2 PlayerStats.SpawnPoint  [private]
```

Definition at line 14 of file PlayerStats.cs.

4.15.4 Property Documentation

4.15.4.1 Instance

```
PlayerStats PlayerStats.Instance  [static], [get], [private set]
```

Definition at line 10 of file PlayerStats.cs.

```
00010 { get; private set; }
```

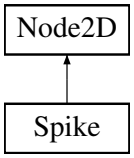
The documentation for this class was generated from the following file:

- PlayerStats.cs

4.16 Spike Class Reference

Klasse für die Spikes.

Inheritance diagram for Spike:



Public Member Functions

- override void _Ready ()

  Initialisierung der Node Player.
- Damage GetDamage ()

  Gibt ein Damage Objekt zurück.

Private Member Functions

- void OnPlayerBodyEntered (Node body)

  Prüfen ob der Körper den Spike betritt falls ja wird der Timer gestartet und der Spieler nimmt Schaden.
- void OnPlayerBodyExited (Node body)

  Prüfen ob der Körper den Spike verlässt, falls ja wird der Timer gestoppt und der Spieler nimmt keinen Schaden mehr.
- void OnTimerTimeout ()

  Timer Timeout Methode, die den Schaden an den Spieler übergibt.

Private Attributes

- Player Player
- float Damage = 10f

4.16.1 Detailed Description

Klasse für die Spikes.

Definition at line 7 of file Spike.cs.

4.16.2 Member Function Documentation

4.16.2.1 _Ready()

```
override void Spike._Ready ()  [inline]
```

Initialisierung der Node Player.

Hier wird der Player Node gefunden

Definition at line 20 of file Spike.cs.

```
00021     {
00022         // Zugriff auf Player Node
00023
00024         Player = GetNode<Player>("../../Player");
00025     }
```

4.16.2.2 GetDamage()

```
Damage Spike.GetDamage ()  [inline]
```

Gibt ein Damage Objekt zurück.

Returns

Damage Objekt

Definition at line 71 of file Spike.cs.

```
00072     {
00073         return new Damage(0, Damage, Vector2.Zero, this);
00074     }
```

4.16.2.3 OnPlayerBodyEntered()

```
void Spike.OnPlayerBodyEntered (
            Node body)  [inline], [private]
```

Prüfen ob der Körper den Spike betritt falls ja wird der Timer gestartet und der Spieler nimmt Schaden.

Definition at line 30 of file Spike.cs.

```
00031     {
00032
00033         if (body is Player)
00034         {
00035             Player = (Player)body; // Instanzvariable setzen
00036             Player.TakeDamage(GetDamage());
00037             Player.SlowPlayer(0.5f);
00038             GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00039             GD.Print("Player entered spike");
00040         }
00041
00042
00043     }
```

4.16.2.4 OnPlayerBodyExited()

```
void Spike.OnPlayerBodyExited (
            Node body)  [inline], [private]
```

Prüfen ob der Körper den Spike verlässt, falls ja wird der Timer gestoppt und der Spieler nimmt keinen Schaden mehr.

Definition at line 48 of file Spike.cs.

```
00049     {
00050         if (body is Player)
00051         {
00052             Player = null; // Instanzvariable zurücksetzen
00053             GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
00054         }
00055     }
```

4.16.2.5 OnTimerTimeout()

```
void Spike.OnTimerTimeout ()  [inline], [private]
```

Timer Timeout Methode, die den Schaden an den Spieler übergibt.

Definition at line 60 of file Spike.cs.

```
00061     {
00062         GD.Print("Timer timeout");
00063         Player.TakeDamage(GetDamage());
00064         GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00065     }
```

4.16.3 Member Data Documentation

4.16.3.1 Damage

```
float Spike.Damage = 10f  [private]
```

Definition at line 14 of file Spike.cs.

4.16.3.2 Player

```
Player Spike.Player  [private]
```

Definition at line 10 of file Spike.cs.

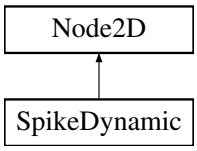
The documentation for this class was generated from the following file:

- Spike.cs

4.17 SpikeDynamic Class Reference

Klasse für die beweglichen Spikes.

Inheritance diagram for SpikeDynamic:



Public Member Functions

- override void _Ready ()

  Initialisierung der Node Player.
- Damage GetDamage ()

  Gibt ein Damage Objekt zurück.

Private Member Functions

- void OnPlayerBodyEntered (Node body)

  Prüfen ob der Körper den Spike betritt falls ja wird der Timer gestartet und der Spieler nimmt Schaden.
- void OnPlayerBodyExited (Node body)

  Prüfen ob der Körper den Spike verlässt, falls ja wird der Timer gestoppt und der Spieler nimmt keinen Schaden mehr.
- void OnTimerTimeout ()

  Timer Timeout Methode, die den Schaden an den Spieler übergibt.

Private Attributes

- Player Player
- float Damage = 10f

4.17.1 Detailed Description

Klasse für die beweglichen Spikes.

Definition at line 7 of file SpikeDynamic.cs.

4.17.2 Member Function Documentation

4.17.2.1 _Ready()

```
override void SpikeDynamic._Ready ()  [inline]
```

Initialisierung der Node Player.

Hier wird der Player Node gefunden

Definition at line 20 of file SpikeDynamic.cs.

```
00021     {
00022         // Zugriff auf Player Node
00023
00024         Player = GetNode<Player>("../../../Player");
00025     }
```

4.17.2.2 GetDamage()

```
Damage SpikeDynamic.GetDamage ()  [inline]
```

Gibt ein Damage Objekt zurück.

Returns

Damage Objekt

Definition at line 71 of file SpikeDynamic.cs.

```
00072     {
00073         return new Damage(0, Damage, Vector2.Zero, this);
00074     }
```

4.17.2.3 OnPlayerBodyEntered()

```
void SpikeDynamic.OnPlayerBodyEntered (
             Node body)  [inline], [private]
```

Prüfen ob der Körper den Spike betritt falls ja wird der Timer gestartet und der Spieler nimmt Schaden.

Definition at line 30 of file SpikeDynamic.cs.

```
00031     {
00032
00033         if (body is Player)
00034         {
00035             Player = (Player)body; // Instanzvariable setzen
00036             Player.TakeDamage(GetDamage());
00037             Player.SlowPlayer(0.5f);
00038             GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00039             GD.Print("Player entered spike");
00040         }
00041
00042
00043     }
```

4.17.2.4 OnPlayerBodyExited()

```
void SpikeDynamic.OnPlayerBodyExited (
             Node body)  [inline], [private]
```

Prüfen ob der Körper den Spike verlässt, falls ja wird der Timer gestoppt und der Spieler nimmt keinen Schaden mehr.

Definition at line 48 of file SpikeDynamic.cs.

```
00049     {
00050         if (body is Player)
00051         {
00052             Player = null; // Instanzvariable zurücksetzen
00053             GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
00054         }
00055     }
```

4.17.2.5 OnTimerTimeout()

```
void SpikeDynamic.OnTimerTimeout ()  [inline], [private]
```

Timer Timeout Methode, die den Schaden an den Spieler übergibt.

Definition at line 60 of file SpikeDynamic.cs.

```
00061     {
00062         GD.Print("Timer timeout");
00063         Player.TakeDamage(GetDamage());
00064         GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00065     }
```

4.17.3 Member Data Documentation

4.17.3.1 Damage

```
float SpikeDynamic.Damage = 10f  [private]
```

Definition at line 13 of file SpikeDynamic.cs.

4.17.3.2 Player

```
Player SpikeDynamic.Player  [private]
```

Definition at line 10 of file SpikeDynamic.cs.

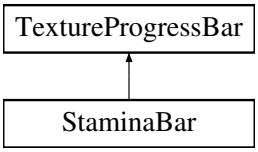
The documentation for this class was generated from the following file:

- SpikeDynamic.cs

4.18 StaminaBar Class Reference

Klasse für die Ausdauerleiste des Spielers. Synchronisiert die Anzeige der StaminaBar mit der Ausdauer des Spielers.

Inheritance diagram for StaminaBar:



Public Member Functions

- override void _Ready ()

  Initialisiert die StaminaBar und verbindet sie mit der Ausdauer des Spielers. Lädt den Spieler-Knoten und setzt die maximale und aktuelle Ausdauer in der StaminaBar.
- override void _Process (double DeltaTime)

  Aktualisiert die StaminaBar in jedem Frame. Synchronisiert die Anzeige der aktuellen Ausdauer mit den Werten des Spielers.

4.18.1 Detailed Description

Klasse für die Ausdauerleiste des Spielers. Synchronisiert die Anzeige der StaminaBar mit der Ausdauer des Spielers.

Definition at line 7 of file StaminaBar.cs.

4.18.2 Member Function Documentation

4.18.2.1 _Process()

```
override void StaminaBar._Process (
            double DeltaTime)  [inline]
```

Aktualisiert die StaminaBar in jedem Frame. Synchronisiert die Anzeige der aktuellen Ausdauer mit den Werten des Spielers.

Parameters

delta	Zeit seit dem letzten Frame (wird nicht direkt genutzt).

Definition at line 24 of file StaminaBar.cs.

```
00024                                                 {
00025         // Aktualisiere den Wert der StaminaBar basierend auf der aktuellen Ausdauer des Spielers
00026         Value = PlayerStats.Instance.GetStamina();
00027     }
```

4.18.2.2 _Ready()

```
override void StaminaBar._Ready ()  [inline]
```

Initialisiert die StaminaBar und verbindet sie mit der Ausdauer des Spielers. Lädt den Spieler-Knoten und setzt die maximale und aktuelle Ausdauer in der StaminaBar.

Definition at line 13 of file StaminaBar.cs.

```
00013                                 {
00014         // Setze die maximale Ausdauer der StaminaBar basierend auf der Spieler-MaxStamina
00015         MaxValue = PlayerStats.Instance.GetMaxStamina();
00016         Value = PlayerStats.Instance.GetStamina();
00017     }
```

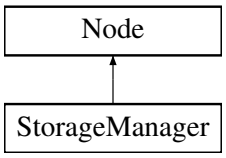
The documentation for this class was generated from the following file:

- StaminaBar.cs

4.19 StorageManager Class Reference

Klasse für das Speichern und Laden von Daten.

Inheritance diagram for StorageManager:



Public Member Functions

- override void _Ready ()

  Initialisierung der Instanz und erstes laden der Einstellungen.
- void LoadSettings ()

  Laden der Einstellungen.
- void LoadGameFile (int id)

  Laden eines Spielstandes.
- void SaveAll (int id)

  Speichern der Einstellungen und eines Spielstandes.
- void SaveSettings ()

  Speichern der Einstellungen.

- void SaveGameFile (int id)

  Speichern eines Spielstandes.
- void SetLastSaveId (int id)

  Setter für LastSaveId.
- int GetLastSaveId ()

  Getter für LastSaveId.
- void SetSaves (int Saves)

  Setter für Saves.
- int GetSaves ()

  Getter für Saves.

Properties

- static StorageManager Instance `[get, private set]`

Private Attributes

- String[ ] PathSave = {"user://save1.dat", "user://save2.dat", "user://save3.dat"}
- int LastSaveId = -1
- int Saves = 0

Static Private Attributes

- const String PathSettings = "user://settings.txt"

4.19.1 Detailed Description

Klasse für das Speichern und Laden von Daten.

Definition at line 8 of file StorageManager.cs.

4.19.2 Member Function Documentation

4.19.2.1 _Ready()

`override void StorageManager._Ready () [inline]`

Initialisierung der Instanz und erstes laden der Einstellungen.

Definition at line 20 of file StorageManager.cs.

```
00020                                {
00021         LoadSettings();
00022         Instance = this;
00023     }
```

4.19.2.2 GetLastSaveId()

```
int StorageManager.GetLastSaveId ()  [inline]
```

Getter für LastSaveId.

Returns

int LastSaveId

Definition at line 118 of file StorageManager.cs.

```
00118                                {
00119         return LastSaveId;
00120     }
```

4.19.2.3 GetSaves()

```
int StorageManager.GetSaves ()  [inline]
```

Getter für Saves.

Returns

int Saves

Definition at line 134 of file StorageManager.cs.

```
00134                           {
00135         return Saves;
00136     }
```

4.19.2.4 LoadGameFile()

```
void StorageManager.LoadGameFile (
             int id)  [inline]
```

Laden eines Spielstandes.

Parameters

Id	des Spielstandes.

Definition at line 43 of file StorageManager.cs.

```
00043                                    {
00044         if(!FileAccess.FileExists(PathSave[id])){
00045             return;
00046         }
00047         FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Read);
00048         PlayerStats.Instance.SetRespawnLevelTag((String) File.GetVar());
00049         PlayerStats.Instance.SetCurrentLevelTag((String) File.GetVar());
00050         PlayerStats.Instance.SetSpawnPoint((Vector2) File.GetVar());
00051         PlayerStats.Instance.SetPosition((Vector2) File.GetVar());
00052         PlayerStats.Instance.SetSinAmount((int) File.GetVar());
00053         PlayerStats.Instance.SetMaxHealthPoints((float) File.GetVar());
00054         PlayerStats.Instance.SetCurrentHealth((float) File.GetVar());
00055         PlayerStats.Instance.SetMaxStamina((float) File.GetVar());
00056         PlayerStats.Instance.SetStamina((float) File.GetVar());
00057         PlayerStats.Instance.SetBVHealAmount((int) File.GetVar());
00058         PlayerStats.Instance.SetBVMaxUses((int) File.GetVar());
00059         PlayerStats.Instance.SetBVCurrentUses((int) File.GetVar());
00060
00061         File.Close();
00062     }
```

4.19.2.5 LoadSettings()

```
void StorageManager.LoadSettings ()  [inline]
```

Laden der Einstellungen.

Definition at line 28 of file StorageManager.cs.

```
00028                               {
00029         if(!FileAccess.FileExists(PathSettings)){
00030             return;
00031         }
00032         FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Read);
00033         Saves = (int) File.GetVar();
00034         LastSaveId = (int) File.GetVar();
00035 
00036         File.Close();
00037     }
```

4.19.2.6 SaveAll()

```
void StorageManager.SaveAll (
            int id)  [inline]
```

Speichern der Einstellungen und eines Spielstandes.

Parameters

Id	des Spielstandes.

Definition at line 68 of file StorageManager.cs.

```
00068                                {
00069         SaveGameFile(id);
00070         SaveSettings();
00071     }
```

4.19.2.7 SaveGameFile()

```
void StorageManager.SaveGameFile (
            int id)  [inline]
```

Speichern eines Spielstandes.

Parameters

Id	des Spielstandes.

Definition at line 88 of file StorageManager.cs.

```
00088                                    {
00089         FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Write);
00090         File.StoreVar(PlayerStats.Instance.GetRespawnLevelTag());
00091         File.StoreVar(PlayerStats.Instance.GetCurrentLevelTag());
00092         File.StoreVar(PlayerStats.Instance.GetSpawnPoint());
00093         File.StoreVar(PlayerStats.Instance.GetPosition());
00094         File.StoreVar(PlayerStats.Instance.GetSinAmount());
00095         File.StoreVar(PlayerStats.Instance.GetMaxHealthPoints());
00096         File.StoreVar(PlayerStats.Instance.GetCurrentHealth());
00097         File.StoreVar(PlayerStats.Instance.GetMaxStamina());
00098         File.StoreVar(PlayerStats.Instance.GetStamina());
00099         File.StoreVar(PlayerStats.Instance.GetBVHealAmount());
00100         File.StoreVar(PlayerStats.Instance.GetBVMaxUses());
00101         File.StoreVar(PlayerStats.Instance.GetBVCurrentUses());
00102 
00103         File.Close();
00104     }
```

4.19.2.8 SaveSettings()

```
void StorageManager.SaveSettings ()  [inline]
```

Speichern der Einstellungen.

Definition at line 76 of file StorageManager.cs.

```
00076                                {
00077         FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Write);
00078         File.StoreVar(Saves);
00079         File.StoreVar(LastSaveId);
00080
00081         File.Close();
00082     }
```

4.19.2.9 SetLastSaveId()

```
void StorageManager.SetLastSaveId (
             int id)  [inline]
```

Setter für LastSaveId.

Parameters

int	LastSaveId

Definition at line 110 of file StorageManager.cs.

```
00110                                        {
00111         LastSaveId = id;
00112     }
```

4.19.2.10 SetSaves()

```
void StorageManager.SetSaves (
             int Saves)  [inline]
```

Setter für Saves.

Parameters

int	Saves

Definition at line 126 of file StorageManager.cs.

```
00126                                    {
00127         this.Saves = Saves;
00128     }
```

4.19.3 Member Data Documentation

4.19.3.1 LastSaveId

```
int StorageManager.LastSaveId = -1  [private]
```

Definition at line 13 of file StorageManager.cs.

4.19.3.2 PathSave

```
String [] StorageManager.PathSave = {"user://save1.dat", "user://save2.dat", "user://save3.↩
dat"}  [private]
```

Definition at line 12 of file StorageManager.cs.
```
00012 {"user://save1.dat", "user://save2.dat", "user://save3.dat"};
```

4.19.3.3 PathSettings

```
const String StorageManager.PathSettings = "user://settings.txt"  [static], [private]
```

Definition at line 11 of file StorageManager.cs.

4.19.3.4 Saves

```
int StorageManager.Saves = 0  [private]
```

Definition at line 14 of file StorageManager.cs.

4.19.4 Property Documentation

4.19.4.1 Instance

```
StorageManager StorageManager.Instance  [static], [get], [private set]
```

Definition at line 10 of file StorageManager.cs.
```
00010 { get; private set; }
```

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

- StorageManager.cs

Chapter 5

File Documentation

5.1 BaseEnemy.cs File Reference

Classes

- class BaseEnemy

  Klasse für einen einfachen Gegner.

5.2 BaseEnemy.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00007 public partial class BaseEnemy : CharacterBody2D
00008 {
00009
00010     private enum State {
00011         IDLE, WALK, ATTACK, TAKE_HIT
00012     }
00013
00014     //customizable variables
00015     [Export]
00016     protected float Damage = 20f;
00017     [Export]
00018     public bool Dead = false;
00019     [Export]
00020     protected bool Respawnable = true;
00021     [Export]
00022     protected float MaxHealthPoints = 100f;
00023     [Export]
00024     public float Armor = 20f; //MUSS ZISCHEN 0 UND 99 LIEGEN
00025     [Export]
00026     protected float MaxStamina = 1f;
00027     [Export]
00028     protected float Speed = 10;
00029     [Export]
00030     protected int SinAmount = 10;
00031     [Export]
00032     protected double ReturnToStartAfter = 5;
00033     [Export(PropertyHint.Flags,
      "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32")]
00034     public uint Id { get; set;} = 0;
00035
00036     //private variables
00037     public float CurrentHealthPoints;
00038     protected float CurrentStamina;
00039     public double ReturnToStart;
00040     public bool Pursuing = false;
00041     protected Node2D CurrentTarget = null;
00042     public Vector2 TargetPosition = Vector2.Inf;
```

```
00043     public Vector2 StartPosition;
00044     protected bool StartRotation = false;
00045     private State AnimationState = State.IDLE;
00046     protected bool AlreadyHit = false;
00047
00048     //linked nodes
00049     public AnimatedSprite2D Sprite;
00050     protected CollisionPolygon2D CollisionPolygon;
00051     protected Area2D SwordHitbox;
00052     public CollisionShape2D MainCollision;
00053     protected RayCast2D FrontCollisionRayCast;
00054     protected RayCast2D LineOfSight;
00055     protected RayCast2D LeftFallProtection;
00056     protected RayCast2D RightFallProtection;
00057     protected TextureProgressBar HealthBar;
00058     protected Player Player;
00059
00064     public override void _Ready()
00065     {
00066         Sprite = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
00067         CollisionPolygon = GetNode<CollisionPolygon2D>("detection/CollisionPolygon2D");
00068         SwordHitbox = GetNode<Area2D>("AnimatedSprite2D/SwordHitBox");
00069         MainCollision = GetNode<CollisionShape2D>("MainCollision");
00070         FrontCollisionRayCast = GetNode<RayCast2D>("FrontCollisionRayCast");
00071         LineOfSight = GetNode<RayCast2D>("LineOfSight");
00072         LeftFallProtection = GetNode<RayCast2D>("LeftFallProtection");
00073         RightFallProtection = GetNode<RayCast2D>("RightFallProtection");
00074         HealthBar = GetNode<TextureProgressBar>("HealthBar");
00075         Player = GetNode<Player>("../../Player");
00076
00077         CurrentHealthPoints = MaxHealthPoints;
00078         CurrentStamina = MaxStamina;
00079         ReturnToStart = ReturnToStartAfter;
00080         StartPosition = Position;
00081         StartRotation = Sprite.FlipH;
00082
00083         HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
00084     }
00085
00091     public override void _Process(double DeltaTime)
00092     {
00093         HandleMovement(DeltaTime);
00094         if(CurrentStamina < MaxStamina){
00095             CurrentStamina += (float) DeltaTime;
00096             Velocity = Velocity * 0.8f;
00097         }
00098         if (!IsOnFloor() && !Dead) {
00099             Velocity += GetGravity() * (float)DeltaTime;
00100         }
00101         UpdateAnimation();
00102         MoveAndSlide();
00103         CheckPlayerHit();
00104     }
00105
00110     public void OnDetectionBodyEntered(Node2D body){
00111         if(CheckLineOfSight(body)){
00112             Pursuing = true;
00113             CurrentTarget = body;
00114         }
00115     }
00116
00121     public void OnPursuingRadiusBodyExited(Node2D body){
00122         if(body == CurrentTarget){
00123             Pursuing = false;
00124             CurrentTarget = null;
00125         }
00126     }
00127
00132     public void OnHitboxAreaEntered(Area2D area){
00133         Player Player1 = (Player) area.GetParent().GetParent();
00134         TakeDamage(Player1.GetDamage());
00135     }
00136
00141     public void OnSwordHitBoxBodyEntered(Node2D body){
00142         if(Dead) return;
00143         Sprite.Play("attack");
00144     }
00145
00150     private void HandleMovement(double DeltaTime){
00151         if(Dead) return;
00152         if((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying()){
00153             Velocity = Vector2.Zero;
00154             return;
00155         }
00156         if(Pursuing){
00157             AnimationState = State.WALK;
00158             TargetPosition = CurrentTarget.Position;
```

```
00159             if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
00160                 AnimationState = State.IDLE;
00161                 Velocity = Vector2.Zero;
00162                 return;
00163             }
00164             ReturnToStart = ReturnToStartAfter;
00165         } else if(ReturnToStart >= 0){
00166             AnimationState = State.IDLE;
00167             ReturnToStart -= DeltaTime;
00168             TargetPosition = Vector2.Inf;
00169         } else if(!IsCloseTo(Position.X, StartPosition.X, 0.1f)){
00170             AnimationState = State.WALK;
00171             TargetPosition = StartPosition;
00172         }
00173 
00174         if(TargetPosition != Vector2.Inf){
00175 
00176             if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
00177                 AnimationState = State.IDLE;
00178                 Velocity = Vector2.Zero;
00179                 if(TargetPosition == StartPosition && Sprite.FlipH != StartRotation){
00180                     FlipRotation();
00181                 }
00182                 TargetPosition = Vector2.Inf;
00183                 return;
00184             }
00185 
00186             if(TargetPosition.X > Position.X){
00187                 SetRotation(true);
00188                 if(!FrontCollisionRayCast.IsColliding()){
00189                     Vector2 velocity = Vector2.Zero;
00190                     velocity.X = Speed;
00191                     Velocity = velocity;
00192                 }
00193             } else {
00194                 SetRotation(false);
00195                 if(!FrontCollisionRayCast.IsColliding()){
00196                     Vector2 velocity = Vector2.Zero;
00197                     velocity.X = -Speed;
00198                     Velocity = velocity;
00199                 }
00200             }
00201 
00202             if((!RightFallProtection.IsColliding() && !Sprite.FlipH) ||
      (!LeftFallProtection.IsColliding() && Sprite.FlipH)){
00203                 Velocity = Vector2.Zero;
00204             }
00205 
00206         } else {
00207             Velocity = Vector2.Zero;
00208             AnimationState = State.IDLE;
00209         }
00210     }
00211 
00212 
00216     protected virtual void UpdateAnimation(){
00217         if(Dead) return;
00218         if(!((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying())){
00219             switch(AnimationState){
00220                 case State.IDLE:
00221                     Sprite.Play("idle");
00222                     break;
00223 
00224                 case State.WALK:
00225                     Sprite.Play("walk");
00226                     break;
00227 
00228                 case State.ATTACK:
00229                     Sprite.Play("attack");
00230                     break;
00231 
00232                 case State.TAKE_HIT:
00233                     Sprite.Play("take_hit");
00234                     break;
00235             }
00236         }
00237         HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
00238 
00239     }
00240 
00245     public void TakeDamage(Damage DMG){
00246         if(Dead) {
00247             return;
00248         }
00249         CurrentHealthPoints -= DMG.GetPhysicalDMG() * (1 - Armor / 100.0f) + DMG.GetTrueDMG();
00250         Position += DMG.GetPushAmount();
00251         if(CurrentHealthPoints <= 0){
```

```
00252             Die();
00253         } else {
00254             Sprite.Play("take_hit");
00255             if(DMG.GetSource() == Player){
00256                 Pursuing = true;
00257                 CurrentTarget = Player;
00258             }
00259         }
00260     }
00261
00266     public bool IsDead(){
00267         return Dead;
00268     }
00269
00274     private void CheckPlayerHit(){
00275         if(Dead) return;
00276         if(Sprite.Animation != "attack"){
00277             AlreadyHit = false;
00278             if(Sprite.Animation == "take_hit" || CurrentStamina < MaxStamina) return;
00279             Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
00280             foreach(Node2D Body in Bodies){
00281                 if(Body == Player){
00282                     Sprite.Play("attack");
00283                 }
00284             }
00285             return;
00286         }
00287         if(AlreadyHit) return;
00288         if(Sprite.Frame >= 6){
00289             CurrentStamina = 0;
00290             Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
00291             foreach(Node2D Body in Bodies){
00292                 if(Body == Player){
00293                     Player.TakeDamage(new Damage(Damage, 0f, Vector2.Zero, this));
00294                     AlreadyHit = true;
00295                     break;
00296                 }
00297             }
00298         }
00299
00300     }
00301
00305     public void Die(){
00306         Dead = true;
00307         Velocity = Vector2.Zero;
00308         MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, true);
00309
00310         Sprite.Play("death");
00311         HealthBar.SetVisible(false);
00312         if (Player != null) {
00313             Player.SetSinAmount(PlayerStats.Instance.GetSinAmount() + SinAmount);
00314         }
00315
00316     }
00317
00321     public void Respawn()
00322     {
00323         Dead = false;
00324         CurrentHealthPoints = MaxHealthPoints;
00325         HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00326         MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, false);
00327         HealthBar.SetVisible(true);
00328         Sprite.Play("idle");
00329     }
00330
00336     private bool CheckLineOfSight(Node2D body){
00337         Vector2 offset = Vector2.Zero;
00338         offset.Y = -14;
00339         LineOfSight.TargetPosition = body.Position + offset - (Position + LineOfSight.Position);
00340         if(LineOfSight.IsColliding()){
00341             return LineOfSight.GetCollider() == body;
00342         }
00343         return true;
00344     }
00345
00349     private void FlipRotation(){
00350         Sprite.FlipH = !Sprite.FlipH;
00351         CollisionPolygon.RotationDegrees = Math.Abs(CollisionPolygon.RotationDegrees -180);
00352         SwordHitbox.RotationDegrees = Math.Abs(SwordHitbox.RotationDegrees -180);
00353         FrontCollisionRayCast.RotationDegrees = Math.Abs(FrontCollisionRayCast.RotationDegrees - 180);
00354     }
00355
00360     private void SetRotation(bool Rotation){
00361         Sprite.FlipH = Rotation ^ StartRotation; // XOR mit StartRotation
00362         if(Rotation){
00363             CollisionPolygon.RotationDegrees = 180;
00364             SwordHitbox.RotationDegrees = 180;
```

```
00365             FrontCollisionRayCast.RotationDegrees = 180;
00366         } else {
00367             CollisionPolygon.RotationDegrees = 0;
00368             SwordHitbox.RotationDegrees = 0;
00369             FrontCollisionRayCast.RotationDegrees = 0;
00370         }
00371     }
00372
00380     private bool IsCloseTo(float Value1, float Value2, float Delta){
00381         return Value1 <= (Value2 + Delta) && Value1 >= (Value2 - Delta);
00382     }
00383 }
```

5.3 BloodVial.cs File Reference

Classes

- class BloodVial

  Klasse für die Interaktion zum heilen.

5.4 BloodVial.cs

Go to the documentation of this file.

```
00001 using GdMUT;
00002 using Godot;
00003 using System;
00004
00008 public partial class BloodVial : Label {
00009
00014     public override void _Ready() {
00015         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00016     }
00017
00021     public void UseBloodVial(){
00022         if(PlayerStats.Instance.GetBVCurrentUses() <= 0) return;
00023         PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVCurrentUses() - 1);
00024         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00025         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() +
      PlayerStats.Instance.GetBVHealAmount());
00026     }
00027
00031     public void ResetUses(){
00032         PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVMaxUses());
00033         Text = PlayerStats.Instance.GetBVCurrentUses() + "";
00034     }
00035
00040     public void AddMaxUses(int Amount){
00041         PlayerStats.Instance.SetBVMaxUses(PlayerStats.Instance.GetBVMaxUses() + Amount);
00042         ResetUses();
00043     }
00044
00048     public void LevelHealAmount(){
00049         PlayerStats.Instance.SetBVHealAmount(PlayerStats.Instance.GetBVHealAmount() + 25);
00050     }
00051 }
```

5.5 Boss1.cs File Reference

Classes

- class Boss1

  Klasse für einen stärkeren Boss-Gegner, der von BaseEnemy erbt.

5.6 Boss1.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00007 public partial class Boss1 : BaseEnemy{
00008
00009     public bool EnemiesRevived = false;
00010     private float RegenCooldown = 5.0f; // Zeit, nach der Regeneration beginnt, wenn kein Schaden
      genommen wurde
00011     private float RegenTimer = 0.0f; // Timer für die Zeit seit dem letzten Angriff
00012     private float RegenAmount = 10.0f; // Menge an Gesundheit, die pro Tick regeneriert wird
00013
00014
00018     public override void _Ready(){
00019
00020         MaxHealthPoints = 400f;
00021         Damage = 50f;
00022         Armor = 30f;
00023         Speed = 10f;
00024         SinAmount = 100; // Bonuspunkte für Spieler beim Besiegen des Bosses
00025
00026         base._Ready();
00027
00028         CurrentHealthPoints = MaxHealthPoints;
00029         HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00030     }
00031
00036     public override void _Process(double DeltaTime){
00037         base._Process(DeltaTime);
00038
00039         if (CurrentHealthPoints <= MaxHealthPoints / 2 && !EnemiesRevived){
00040             StartGlowing();
00041             ReviveEnemies();
00042             EnemiesRevived = true;
00043             Armor = 60f; // Rüstung erhöhen
00044         }
00045
00046         HandleRegeneration(DeltaTime);
00047     }
00048
00053     public void HandleRegeneration(double DeltaTime){
00054         if (CurrentHealthPoints < MaxHealthPoints){
00055             RegenTimer += (float)DeltaTime;
00056
00057             if (RegenTimer >= RegenCooldown){
00058                 CurrentHealthPoints = Math.Min(CurrentHealthPoints + RegenAmount, MaxHealthPoints);
00059                 HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
00060                 RegenTimer = 0.0f; // Timer zurücksetzen
00061             }
00062         }
00063     }
00064
00068     public void StartGlowing(){
00069         // Ändere die Modulationsfarbe des Sprites, um ein Leuchten zu simulieren
00070         if (Sprite != null){
00071             ShowPopupMessage("AHHHH!!!");
00072             Sprite.Modulate = new Color(1.0f, 0.84f, 0.0f, 1.0f); // Ein goldliche Leuchteffekt
00073         }
00074     }
00075
00080     private void ShowPopupMessage(string Message){
00081         Label popup = new Label();
00082         popup.Text = Message;
00083         popup.AddThemeColorOverride("font_color", new Color(1, 0, 0)); // Rot
00084         popup.Modulate = new Color(1, 1, 1, 0); // Start transparent
00085         popup.AutowrapMode = TextServer.AutowrapMode.Word;
00086         popup.SizeFlagsHorizontal = (Control.SizeFlags)(int)Control.SizeFlags.ExpandFill;
00087         popup.SizeFlagsVertical = (Control.SizeFlags)(int)Control.SizeFlags.ShrinkCenter;
00088         popup.HorizontalAlignment = HorizontalAlignment.Center;
00089         popup.VerticalAlignment = VerticalAlignment.Center;
00090
00091
00092         Vector2 bossGlobalPosition = GetGlobalTransformWithCanvas().Origin;
00093         popup.GlobalPosition = bossGlobalPosition + new Vector2(0, -100);
00094
00095         CanvasLayer canvas = new CanvasLayer();
00096         AddChild(canvas);
00097         canvas.AddChild(popup);
00098
00099         var tween = CreateTween();
00100         tween.TweenProperty(popup, "modulate:a", 1, 0.5f).From(0); // Einblenden
00101         tween.TweenProperty(popup, "modulate:a", 0, 0.5f).From(1).SetDelay(1.0f); // Ausblenden nach 1
      Sekunde
```

```
00102         tween.Connect("finished", new Callable(popup, "queue_free"));
00103     }
00104
00108     private void ReviveEnemies()
00109     {
00110         // Hole den Elternknoten (bossRoom)
00111         Node BossRoom = GetParent();
00112
00113         // Iteriere durch alle Kinder von bossRoom
00114         foreach (Node Child in BossRoom.GetChildren()){
00115             if (Child is BaseEnemy BaseEnemy && BaseEnemy.IsDead()){
00116                 BaseEnemy.Respawn();
00117             }
00118         }
00119     }
00120 }
```

5.7 Checkpoint.cs File Reference

Classes

- class Checkpoint

5.8 Checkpoint.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00004 public partial class Checkpoint : Node2D
00005 {
00006
00007     // Variable für Player
00008     private Player Player;
00009
00010     /*
00011      * @brief Intitalisierung der Node Player
00012      */
00013     public override void _Ready()
00014     {
00015         // Zugriff auf Player Node
00016         Player = GetNode<Player>("../Player");
00017     }
00018
00019     /*
00020      * @brief Diese Funktion wird aufgerufen, wenn der Player den Checkpoint betritt
00021      * @param body Der Körper, der den Checkpoint betritt
00022      */
00023     private void OnPlayerBodyEntered(Node body)
00024     {
00025
00031         if (body is Player Player)
00032         {
00033             // Setzen des Spawnpoints
00034             PlayerStats PlayerStats = GetNode<PlayerStats>("/root/PlayerStats");
00035             PlayerStats.Instance.SetSpawnPoint(this.GlobalPosition);
00036             Player.MaxHeal();
00037             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetMaxStamina());
00038             Player.GetBloodVials().ResetUses();
00039             GD.Print("Spawnpoint des Players gesetzt auf: ", this.GlobalPosition);
00040
00041             PlayerStats.SetRespawnLevelTag(GetParent().Name);
00042             GD.Print("RespawnLevelTag des Players gesetzt auf: ", GetParent().Name);
00043             GD.Print(PlayerStats.Instance.GetRespawnLevelTag());
00044         }
00045
00046     }
00047 }
```

5.9 Damage.cs File Reference

Classes

- class Damage

  Repräsentiert den Schaden, der von Charakteren oder Gegnern verursacht wird. Beinhaltet physischen Schaden, wahren Schaden und den Rückstoßeffekt.

5.10 Damage.cs

Go to the documentation of this file.

```
00001 using Godot;
00002
00007 public class Damage{
00008
00009     private float PhysicalDMG;
00010     private float TrueDMG;
00011     private Vector2 PushAmount;
00012     private Node2D Source;
00013
00020     public Damage(float PhysicalDMG, float TrueDMG, Vector2 PushAmount, Node2D Source){
00021         this.PhysicalDMG = PhysicalDMG;
00022         this.TrueDMG = TrueDMG;
00023         this.PushAmount = PushAmount;
00024         this.Source = Source;
00025     }
00026
00031     public float GetPhysicalDMG(){
00032         return PhysicalDMG;
00033     }
00034
00039     public float GetTrueDMG(){
00040         return TrueDMG;
00041     }
00042
00047     public Vector2 GetPushAmount(){
00048         return PushAmount;
00049     }
00050
00055     public Node2D GetSource(){
00056         return Source;
00057     }
00058 }
```

5.11 Door.cs File Reference

Classes

- class Door

  Klasse für die Tür.

5.12 Door.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00008 public partial class Door : Area2D
00009 {
00010     public Node Spawn;
00011
00012     [Export]
00013     public string DestinationLevelTag { get; set; }
```

```
00014
00015     [Export]
00016     public string DestinationDoorTag { get; set; }
00017
00018     [Export]
00019     public string SpawnDirection { get; set; } = "up";
00020
00021
00022
00026     public override void _Ready()
00027     {
00028         Spawn = GetNode("Spawn");
00029     }
00030
00031
00036     private void OnPlayerBodyEntered(Node body)
00037     {
00038         if (body is Player player)
00039         {
00040             var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00041             NavigationManager.GoToLevel(DestinationLevelTag, DestinationDoorTag);
00042         }
00043     }
00044 }
```

5.13 HealthBar.cs File Reference

Classes

- class HealthBar

  Klasse für die Gesundheitsleiste des Spielers. Synchronisiert die Anzeige der HealthBar mit den Lebenspunkten des Spielers.

5.14 HealthBar.cs

Go to the documentation of this file.

```
00001 using Godot;
00002
00007 public partial class HealthBar : TextureProgressBar {
00008
00013     public override void _Ready() {
00014         // Setze die maximale Gesundheit der HealthBar basierend auf der Spieler-MaxHealth
00015         MaxValue = PlayerStats.Instance.GetMaxHealthPoints();
00016         Value = PlayerStats.Instance.GetCurrentHealth();
00017     }
00018
00024     public override void _Process(double DeltaTime) {
00025         // Aktualisiere den Wert der HealthBar basierend auf der aktuellen Gesundheit des Spielers
00026         Value = PlayerStats.Instance.GetCurrentHealth();
00027     }
00028 }
```

5.15 Hud.cs File Reference

Classes

- class Hud

  Klasse für das PauseMenu.

5.16 Hud.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00004
00008 public partial class Hud : CanvasLayer {
00009
00010     private AnimationPlayer AnimationPlayer;
00011     private CenterContainer Buttons;
00012     private bool Enabled;
00013
00014
00019     public override void _Ready() {
00020         AnimationPlayer = GetNode<AnimationPlayer>("PauseMenu/AnimationPlayer");
00021         Buttons = GetNode<CenterContainer>("PauseMenu/Buttons");
00022         AnimationPlayer.Play("RESET");
00023     }
00024
00029     public override void _Process(double DeltaTime) {
00030         if(Input.IsActionJustPressed("escape")){
00031             TogglePause();
00032         }
00033     }
00034
00038     private void TogglePause(){
00039         Enabled = !Enabled;
00040         GetTree().Paused = Enabled;
00041         if(Enabled){
00042             AnimationPlayer.Play("Pause");
00043             Buttons.Visible = true;
00044         } else {
00045             AnimationPlayer.PlayBackwards("Pause");
00046             Buttons.Visible = false;
00047         }
00048     }
00049
00053     public void OnResumeButtonPressed(){
00054         TogglePause();
00055     }
00056
00060     public void OnSaveButtonPressed(){
00061         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00062     }
00063
00067     public void OnSaveMenuButtonPressed(){
00068         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00069         NavigationManager.Instance.GoToLevel("main_menu", null);
00070         PlayerStats.Instance.Reload();
00071         GetTree().Paused = false;
00072     }
00073
00077     public void OnSaveQuitButtonPressed(){
00078         StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
00079         GetTree().Quit();
00080     }
00081
00082 }
```

5.17 Interactable.cs File Reference

Classes

- class Interactable

  Klasse für Interaktion.

5.18 Interactable.cs

Go to the documentation of this file.

```
00001 using Godot;
```

```
00002 using System;
00003
00007 public partial class Interactable : AnimatedSprite2D {
00008
00009     private Player Player;
00010     private RichTextLabel TextLabel;
00011     private Control PopUp;
00012     private Area2D Area;
00013
00014     [Export(PropertyHint.MultilineText)]
00015     private String Text { get; set;}
00016
00021     public override void _Ready(){
00022         Player = GetNode<Player>("../Player");
00023         TextLabel = GetNode<RichTextLabel>("../HUD/PopUp/Text");
00024         PopUp = GetNode<Control>("../HUD/PopUp");
00025         Area = GetNode<Area2D>("Area2D");
00026     }
00027
00032     public override void _Process(double DeltaTime){
00033         if(Input.IsActionJustPressed("interact")){
00034             Godot.Collections.Array<Node2D> Bodies = Area.GetOverlappingBodies();
00035             foreach(Node2D Body in Bodies){
00036                 if(Body == Player){
00037                     TextLabel.Clear();
00038                     TextLabel.AppendText(Text);
00039                     PopUp.Visible = true;
00040                     return;
00041                 }
00042             }
00043         }
00044     }
00045
00050     public void OnAreaBodyExited(Node2D Body){
00051         if(Body == Player){
00052             PopUp.Visible = false;
00053             TextLabel.Clear();
00054         }
00055     }
00056
00057 }
```

5.19 LevelManager.cs File Reference

Classes

- class LevelManager

  Klasse für den LevelManager Diese Klasse verwaltet den Levelwechsel und die Spielerpositionierung.

5.20 LevelManager.cs

Go to the documentation of this file.

```
00001 using Godot;
00002
00007 public partial class LevelManager : Node2D
00008 {
00013     public override void _Ready()
00014     {
00015         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00016
00021         if (NavigationManager.SpawnDoorTag != null)
00022         {
00023             OnLevelSpawn(NavigationManager.SpawnDoorTag);
00024         }
00025         else
00026         {
00027             NavigationManager.CallDeferred("TriggerPlayerSpawn", PlayerStats.Instance.GetPosition(),
      "");
00028         }
00029
00030     }
00031
00036     private void OnLevelSpawn(string DestinationTag)
```

```
00037     {
00038         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00039         // Pfad zur Tür basierend auf dem Ziel-Tag erstellen
00040         string DoorPath = "Doors/Door_" + DestinationTag;
00041 
00042         Door door = GetNode<Door>(DoorPath);
00043 
00044         // TriggerPlayerSpawn nach deferred ausführen
00045         NavigationManager.CallDeferred("TriggerPlayerSpawn", door.GlobalPosition,
      door.SpawnDirection);
00046     }
00047 }
```

5.21 MainMenu.cs File Reference

Classes

- class MainMenu

  Klasse für das MainMenu.

5.22 MainMenu.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003 
00007 public partial class MainMenu : Node2D {
00008 
00009     private int MenuState = 0;
00010     private VBoxContainer Navigation;
00011     private MarginContainer SavesContainer;
00012     private Button ContinueButton;
00013     private Label InfoLabel;
00014     private Label[] SaveLabel = new Label[3];
00015     private Button[] SelectButton = new Button[3];
00016     private Button[] DeleteButton = new Button[3];
00017     private ConfirmationDialog DeleteConfirmation;
00018     private int SaveToDelete = 0;
00019 
00020 
00025     public override void _Ready() {
00026         Navigation = GetNode<VBoxContainer>("Control/Navigation");
00027         SavesContainer = GetNode<MarginContainer>("Control/Saves");
00028         ContinueButton = GetNode<Button>("Control/Navigation/ContinueButton");
00029         InfoLabel = GetNode<Label>("Control/Saves/VBoxContainer/Info");
00030 
00031         SaveLabel[0] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Label");
00032         SelectButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Select");
00033         DeleteButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Delete");
00034         SaveLabel[1] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Label");
00035         SelectButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Select");
00036         DeleteButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Delete");
00037         SaveLabel[2] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Label");
00038         SelectButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Select");
00039         DeleteButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Delete");
00040 
00041         DeleteConfirmation = GetNode<ConfirmationDialog>("DeleteConfirmation");
00042 
00043         if(StorageManager.Instance.GetLastSaveId() > -1){
00044             ContinueButton.Visible = true;
00045         }
00046     }
00047 
00048 
00052     private void Change(){
00053         if(MenuState == 0){
00054             SavesContainer.Visible = false;
00055             Navigation.Visible = true;
00056         } else {
00057             Navigation.Visible = false;
00058             SavesContainer.Visible = true;
00059 
00060             int Saves = StorageManager.Instance.GetSaves();
```

```
00061
00062             if(MenuState == 1){
00063                 InfoLabel.Text = "Select empty save to start a new Game";
00064                 for(int i = 0; i < 3; i++){
00065                     if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
00066                         SaveLabel[i].Text = "Save " + (i+1);
00067                         SelectButton[i].Disabled = true;
00068                         DeleteButton[i].Disabled = false;
00069                     } else {
00070                         SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
00071                         SelectButton[i].Disabled = false;
00072                         DeleteButton[i].Disabled = true;
00073                     }
00074                 }
00075             } else {
00076                 InfoLabel.Text = "Select save to load Game";
00077                 for(int i = 0; i < 3; i++){
00078                     if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
00079                         SaveLabel[i].Text = "Save " + (i+1);
00080                         SelectButton[i].Disabled = false;
00081                         DeleteButton[i].Disabled = false;
00082                     } else {
00083                         SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
00084                         SelectButton[i].Disabled = true;
00085                         DeleteButton[i].Disabled = true;
00086                     }
00087                 }
00088             }
00089         }
00090     }
00091
00095     public void OnContinueButtonPressed(){
00096         StorageManager.Instance.LoadGameFile(StorageManager.Instance.GetLastSaveId());
00097         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00098     }
00099
00103     public void OnQuitButtonPressed(){
00104         StorageManager.Instance.SaveSettings();
00105         GetTree().Quit();
00106     }
00107
00111     public void OnNewGameButtonPressed(){
00112         MenuState = 1;
00113         Change();
00114     }
00115
00119     public void OnLoadGameButtonPressed(){
00120         MenuState = 2;
00121         Change();
00122     }
00123
00127     public void OnBackButtonPressed(){
00128         MenuState = 0;
00129         Change();
00130     }
00131
00135     public void OnSave1SelectPressed(){
00136         if(MenuState == 2){
00137             StorageManager.Instance.LoadGameFile(0);
00138         }
00139         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00140         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 1);
00141         StorageManager.Instance.SetLastSaveId(0);
00142     }
00143
00147     public void OnSave1DeletePressed(){
00148         SaveToDelete = 1;
00149         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00150         DeleteConfirmation.Show();
00151     }
00152
00156     public void OnSave2SelectPressed(){
00157         if(MenuState == 2){
00158             StorageManager.Instance.LoadGameFile(1);
00159         }
00160         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00161         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 2);
00162         StorageManager.Instance.SetLastSaveId(1);
00163     }
00164
00168     public void OnSave2DeletePressed(){
00169         SaveToDelete = 2;
00170         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00171         DeleteConfirmation.Show();
00172     }
00173
00177     public void OnSave3SelectPressed(){
```

```
00178         if(MenuState == 2){
00179             StorageManager.Instance.LoadGameFile(2);
00180         }
00181         NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
00182         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 4);
00183         StorageManager.Instance.SetLastSaveId(2);
00184     }
00185 
00189     public void OnSave3DeletePressed(){
00190         SaveToDelete = 3;
00191         DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
00192         DeleteConfirmation.Show();
00193     }
00194 
00198     public void OnDeleteConfirmationCanceled(){
00199         SaveToDelete = 0;
00200         Change();
00201     }
00202 
00206     public void OnDeleteConfirmationConfirmed(){
00207         StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() ^ (int) Math.Pow(2,
      SaveToDelete - 1));
00208         Change();
00209     }
00210 
00214     public void OnDeleteConfirmationCloseRequested(){
00215         OnDeleteConfirmationCanceled();
00216     }
00217 
00218 }
```

5.23 MainMenuBackground.cs File Reference

Classes

- class MainMenuBackground

  Klasse für die MainMenuBackground-Animation.

5.24 MainMenuBackground.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003 
00007 public partial class MainMenuBackground : ParallaxLayer {
00008 
00009     [Export]
00010     private float ScrollSpeed = -10f;
00011 
00016     public override void _Process(double DeltaTime) {
00017         float X = GetMotionOffset().X;
00018         X += ScrollSpeed * (float) DeltaTime;
00019         SetMotionOffset(new Vector2(X,0));
00020     }
00021 }
```

5.25 NavigationManager.cs File Reference

Classes

- class NavigationManager

  Der NavigationManager ist für das Laden von Leveln und das Spawnen des Spielers verantwortlich. Der NavigationManager ist ein Singleton, der in der Haupt-Szene platziert wird und von anderen Skripten verwendet wird, um Level zu laden und den Spieler zu spawnen.

5.26 NavigationManager.cs

Go to the documentation of this file.

```
00001 using Godot;
00002
00007 public partial class NavigationManager : Node
00008 {
00009     public static NavigationManager Instance { get; private set; }
00010     // Deklarieren der vorab geladenen Szenen
00011     private static readonly PackedScene SceneMainMenu =
      (PackedScene)GD.Load("res://Scenes/main_menu.tscn");
00012     private static readonly PackedScene SceneIntro = (PackedScene)GD.Load("res://Scenes/intro.tscn");
00013     private static readonly PackedScene SceneLevel1 =
      (PackedScene)GD.Load("res://Scenes/level1.tscn");
00014     private static readonly PackedScene SceneBoss =
      (PackedScene)GD.Load("res://Scenes/bossRoom.tscn");
00015     private static readonly PackedScene SceneLevelOne =
      (PackedScene)GD.Load("res://Scenes/level_one.tscn");
00016     private static readonly PackedScene SceneLevelTwo =
      (PackedScene)GD.Load("res://Scenes/level_two.tscn");
00017
00018     // Die Spawn-Tag-Variable
00019     public string SpawnDoorTag { get; private set; }
00020
00026     [Signal]
00027     public delegate void OnTriggerPlayerSpawnEventHandler(Vector2 Position, string Direction);
00028
00032     public override void _Ready(){
00033         Instance = this;
00034     }
00035
00041     public void GoToLevel(string LevelTag, string DestinationTag)
00042     {
00043         PackedScene SceneToLoad = null;
00044
00045         // Bestimmen, welches Level geladen werden soll
00046         switch (LevelTag)
00047         {
00048             case "main_menu":
00049                 SceneToLoad = SceneMainMenu;
00050                 break;
00051             case "intro":
00052                 SceneToLoad = SceneIntro;
00053                 break;
00054             case "level1":
00055                 SceneToLoad = SceneLevel1;
00056                 break;
00057             case "bossRoom":
00058                 SceneToLoad = SceneBoss;
00059                 break;
00060             case "level_one":
00061                 SceneToLoad = SceneLevelOne;
00062                 break;
00063             case "level_two":
00064                 SceneToLoad = SceneLevelTwo;
00065                 break;
00066         }
00067
00068         // Überprüfen, ob eine Szene ausgewählt wurde und diese dann laden
00069         if (SceneToLoad != null){
00070             if(SceneToLoad != SceneMainMenu){
00071                 PlayerStats.Instance.SetCurrentLevelTag(LevelTag);
00072                 SpawnDoorTag = DestinationTag;
00073             }
00074             // Verwendung der ChangeSceneToPacked-Methode in Godot 4
00075             CallDeferred(nameof(DeferredChangeScene), SceneToLoad);
00076         }
00077     }
00078
00083     private void DeferredChangeScene(PackedScene SceneToLoad)
00084     {
00085         GetTree().ChangeSceneToPacked(SceneToLoad);
00086     }
00087
00093     public void TriggerPlayerSpawn(Vector2 Position, string Direction)
00094     {
00095         EmitSignal(SignalName.OnTriggerPlayerSpawn, Position, Direction);
00096     }
00097 }
```

5.27 Player.cs File Reference

Classes

- class Player

  Klasse für den Spielercharakter. Verwaltet Bewegung, Sprünge, Angriffe und Animationen.

5.28 Player.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00008 public partial class Player : CharacterBody2D
00009 {
00010     // Variablen für Bewegung, Sprünge und Dash
00011     private const float SPEED = 100f;
00012     private const float JUMP_VELOCITY = -300f;
00013     private int JumpMax = 2;
00014     private int JumpCount = 0;
00015
00016     private Vector2 DashDirection = Vector2.Zero;
00017     private float DashSpeed = 300f;
00018     private bool IsDashing = false;
00019     private bool CanDash = true;
00020     private float DashTrailInterval = 0.05f;
00021     private float DashTrailTimer = 0f;
00022
00023     // Referenzen zu den Knoten
00024     private AnimationPlayer AnimationPlayer;
00025     private Sprite2D Sprite;
00026     private Timer DashEffect;
00027     private Timer DashTimer;
00028     private CollisionShape2D SwordCollision;
00029     private CollisionShape2D PlayerHitbox;
00030     private BloodVial BloodVials;
00031     private Label SinDisplay;
00032
00033     private Vector2 HauptHitbox;
00034     private int LastAttack = 0;
00035
00036     //Variablen für Stamina
00037     private float TimeSinceLastStaminaUse = 0f;
00038
00043     public override void _Ready() {
00044         AnimationPlayer = GetNode<AnimationPlayer>("AnimationPlayer");
00045         Sprite = GetNode<Sprite2D>("Sprite2D");
00046         DashEffect = GetNode<Timer>("DashEffect");
00047         DashTimer = GetNode<Timer>("DashTimer");
00048         SwordCollision = GetNode<CollisionShape2D>("Sprite2D/SwordHit/SwordCollision");
00049         PlayerHitbox = GetNode<CollisionShape2D>("PlayerHitbox");
00050         HauptHitbox = PlayerHitbox.Position;
00051         BloodVials = GetNode<BloodVial>("../HUD/BloodVial/Counter");
00052         SinDisplay = GetNode<Label>("../HUD/SinAmount/Counter");
00053
00054         SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";
00055
00056         NavigationManager navigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00057         navigationManager.Connect("OnTriggerPlayerSpawn", new Callable(this, nameof(OnSpawn)));
00058
00059         Position = PlayerStats.Instance.GetPosition();
00060     }
00061
00067     public override void _PhysicsProcess(double DeltaTime) {
00068         // Gravitation hinzufügen, wenn der Charakter nicht am Boden ist
00069         if (!IsOnFloor()) {
00070             Velocity += GetGravity() * (float)DeltaTime;
00071         } else {
00072             CanDash = true; // Dash wird zurückgesetzt, wenn der Charakter am Boden ist
00073         }
00074
00075         TimeSinceLastStaminaUse += (float)DeltaTime;
00076         RegenerateStamina(20f, DeltaTime);
00077
00078         // Heal
00079         if(Input.IsActionJustPressed("heal")){
00080             BloodVials.UseBloodVial();
```

```
00081         }
00082
00083         HandleJump();
00084         HandleMovement(DeltaTime);
00085         MoveAndSlide();
00086         UpdateAnimations();
00087         PlayerStats.Instance.SetPosition(Position);
00088         }
00089
00094     private void HandleJump() {
00095         // Sprungzähler zurücksetzen, wenn der Charakter am Boden ist
00096         if (JumpCount != 0 && IsOnFloor()) {
00097             JumpCount = 0;
00098         }
00099
00100         // Überprüfen, ob der Sprung-Button gedrückt wurde und der Charakter noch Sprünge übrig hat
00101         if (Input.IsActionJustPressed("ui_up") && JumpCount < JumpMax) {
00102             if (JumpCount == 0) {
00103             // Erster Sprung ohne Stamina-Verlust
00104             Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
00105             JumpCount += 1;
00106             } else if (JumpCount > 0) {
00107                 // Beim Doppelsprung Stamina prüfen und abziehen
00108                 if (UseStamina(15)) {
00109                     Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
00110                     JumpCount += 1;
00111                 }
00112             }
00113         }
00114     }
00115
00121     private void HandleMovement(double DeltaTime) {
00122         Vector2 direction = new Vector2(Input.GetAxis("ui_left", "ui_right"), Input.GetAxis("ui_up",
      "ui_down")).Normalized();
00123         float currentSpeed = SPEED;
00124
00125         // Sprite umdrehen basierend auf der Bewegungsrichtung und Kollision umdrehen
00126         if (direction.X < 0) {
00127             Sprite.FlipH = true;
00128             SwordCollision.Position = new Vector2(-Mathf.Abs(SwordCollision.Position.X),
      SwordCollision.Position.Y);
00129             PlayerHitbox.Position = new Vector2(Sprite.Position.X * 1.8f, PlayerHitbox.Position.Y);
00130         } else if (direction.X > 0) {
00131             Sprite.FlipH = false;
00132             SwordCollision.Position = new Vector2(Mathf.Abs(SwordCollision.Position.X),
      SwordCollision.Position.Y);
00133             PlayerHitbox.Position = HauptHitbox;
00134         }
00135
00136         // Geschwindigkeit reduzieren, wenn der Spieler angreift
00137         if (AnimationPlayer.CurrentAnimation == "light_attack") {
00138             currentSpeed *= 0.5f;
00139         } else if (AnimationPlayer.CurrentAnimation == "heavy_attack") {
00140             currentSpeed *= 0.15f;
00141         }
00142
00143         // Blockieren stoppt die Bewegung
00144         if (IsBlocking()) {
00145             currentSpeed = 0;
00146         }
00147
00148         if (IsDashing) {
00149             DashInProgress(DeltaTime);
00150         } else {
00151             // Normale Bewegung verarbeiten, wenn kein Dash aktiv ist
00152             if (direction != Vector2.Zero) {
00153                 Velocity = new Vector2(direction.X * currentSpeed, Velocity.Y);
00154             } else {
00155                 Velocity = new Vector2(Mathf.MoveToward(Velocity.X, 0, SPEED), Velocity.Y);
00156             }
00157
00158             // Überprüfen, ob der Dash-Button gedrückt wurde mit eine Bewegungsrichtung und nicht
      schon am angreifen ist
00159             if (Input.IsActionJustPressed("dash") && direction != Vector2.Zero && CanDash &&
      !IsAttacking()) {
00160                 // Wenn der Player genug Stamina hat kann er dashen
00161                 if (UseStamina(20)){
00162                     DashDirection = direction;
00163                     StartDash();
00164                 }
00165             }
00166         }
00167     }
00168
00172     private void StartDash() {
00173         SetCollisionLayerValue(1,false);
00174         SetCollisionMaskValue(1,false);
```

```
00175         IsDashing = true;
00176         CanDash = false;
00177         DashTimer.Timeout += StopDash;
00178         DashTimer.Start();
00179         DashEffect.Start();
00180         DashTrailTimer = 0f;
00181     }
00182
00187     private void DashInProgress(double DeltaTime) {
00188         // Charakter bewegt sich in die Dash-Richtung mit Dash-Geschwindigkeit
00189         if (DashDirection == Vector2.Up) {
00190             Velocity = DashDirection / 1.5f * DashSpeed;
00191         } else {
00192             Velocity = DashDirection * DashSpeed;
00193         }
00194
00195         // Dash-Trail bei Intervallen erstellen
00196         DashTrailTimer -= (float)DeltaTime;
00197         if (DashTrailTimer <= 0f) {
00198             CreateDashEffect();
00199             DashTrailTimer = DashTrailInterval;
00200         }
00201     }
00202
00207     private void CreateDashEffect() {
00208         Sprite2D PlayerCopyNode = (Sprite2D)Sprite.Duplicate();
00209         GetParent().AddChild(PlayerCopyNode);
00210
00211         CollisionShape2D SwordCollisionCopy =
    PlayerCopyNode.GetNode<CollisionShape2D>("SwordHit/SwordCollision");
00212         if (SwordCollisionCopy != null) {
00213             SwordCollisionCopy.Disabled = true; // Deaktiviere die Kollision der Kopie
00214         }
00215
00216         PlayerCopyNode.GlobalPosition = GlobalPosition + new Vector2(0, Sprite.Texture.GetHeight() *
    Sprite.Scale.Y * -0.5f);
00217
00218         // Verblassen-Effekt für den Dash-Trail hinzufügen
00219         float AnimationTime = (float)(DashTimer.WaitTime / 3);
00220
00221         Timer FadeTimer1 = new Timer();
00222         AddChild(FadeTimer1);
00223         FadeTimer1.Timeout += () => {
00224             if (IsInstanceValid(PlayerCopyNode)) {
00225                 PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.4f);
00226             }
00227         };
00228         FadeTimer1.Start(AnimationTime);
00229
00230         Timer FadeTimer2 = new Timer();
00231         AddChild(FadeTimer2);
00232         FadeTimer2.Timeout += () => {
00233             if (IsInstanceValid(PlayerCopyNode)) {
00234                 PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.2f);
00235             }
00236         };
00237         FadeTimer2.Start(AnimationTime * 2);
00238
00239         Timer FadeTimer3 = new Timer();
00240         AddChild(FadeTimer3);
00241         FadeTimer3.Timeout += () => {
00242             if (IsInstanceValid(PlayerCopyNode)) {
00243                 PlayerCopyNode.QueueFree();
00244             }
00245         };
00246         FadeTimer3.Start(AnimationTime * 3);
00247     }
00248
00252     private void StopDash() {
00253         IsDashing = false;
00254         DashEffect.Stop();
00255         DashTimer.Stop();
00256         DashTimer.Timeout -= StopDash;
00257         SetCollisionLayerValue(1,true);
00258         SetCollisionMaskValue(1,true);
00259     }
00260
00265     public bool IsAttacking() {
00266         return AnimationPlayer.CurrentAnimation == "heavy_attack" || AnimationPlayer.CurrentAnimation
    == "light_attack";
00267     }
00268
00273     public bool IsBlocking() {
00274         return AnimationPlayer.CurrentAnimation == "block";
00275     }
00276
00280     public void MaxHeal(){
```

```
00281         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetMaxHealthPoints());
00282     }
00283
00289     public void TakeDamage(Damage Damage){
00290         float totalDamage = Damage.GetTrueDMG();
00291         if(!IsBlocking()){
00292             totalDamage += Damage.GetPhysicalDMG();
00293         } else {
00294             float CurrentStamina = PlayerStats.Instance.GetStamina();
00295             CurrentStamina -= Damage.GetPhysicalDMG();
00296             if(CurrentStamina < 0){
00297                 totalDamage -= CurrentStamina;
00298             }
00299             PlayerStats.Instance.SetStamina(CurrentStamina);
00300         }
00301
00302         PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() - totalDamage);
00303         Position += Damage.GetPushAmount();
00304
00305         // Überprüfe, ob der Spieler gestorben ist
00306         if (PlayerStats.Instance.GetCurrentHealth() <= 0){
00307             GD.Print("Spieler ist gestorben!");
00308             Respawn();
00309         }
00310     }
00311
00317     public Damage GetDamage(){
00318         if(LastAttack == 1){
00319             return new Damage(50, 0, Vector2.Zero, this);
00320         }
00321         if(LastAttack == 2){
00322             Vector2 Push = new Vector2(20,0);
00323             if(Sprite.FlipH){
00324                 Push = -Push;
00325             }
00326             return new Damage(100, 0, Push, this);
00327         }
00328         return new Damage(0,0,Vector2.Zero, this);
00329     }
00330
00336     public void RegenerateStamina(float Amount, double delta) {
00337         // Wenn die Verzögerungszeit erreicht wurde, regeneriere Stamina
00338         if (TimeSinceLastStaminaUse >= 1f) {
00339             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() + Amount *
      (float)delta); // Regeneriere Stamina abhängig von der Zeit
00340         }
00341     }
00342
00349     public bool UseStamina(float Amount) {
00350         // Versucht, eine bestimmte Menge an Stamina zu verbrauchen.
00351         // Gibt true zurück, wenn genug Stamina verfügbar war; andernfalls false.
00352         if (PlayerStats.Instance.GetStamina() >= Amount) {
00353             PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() - Amount);
00354             TimeSinceLastStaminaUse = 0f;
00355             return true;
00356         }
00357
00358         return false;
00359     }
00360
00365     public void SlowPlayer(float SlowAmount){
00366         Velocity = new Vector2(Velocity.X * SlowAmount, Velocity.Y);
00367     }
00368
00372     public void Respawn(){
00373         var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
00374         NavigationManager.GoToLevel(PlayerStats.Instance.GetRespawnLevelTag(), "spawn");
00375         BloodVials.ResetUses();
00376
00377     }
00378
00383     public BloodVial GetBloodVials(){
00384         return BloodVials;
00385     }
00386
00391     public void SetSinAmount(int Value) {
00392         // SinAmount muss immer >= 0 sein
00393         PlayerStats.Instance.SetSinAmount(Value);
00394         SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";
00395     }
00396
00402     private void OnSpawn(Vector2 position, string direction){
00403
00404         // Spielerposition auf die übergebene Position setzen
00405         if (direction == "right")
00406         {
00407             // Update the x value by adding 50 to it, keep the original y value
```

```
00408             Sprite.FlipH = false;
00409             position = position with { X = position.X + 25 };
00410         }
00411         else if (direction == "left")
00412         {
00413             // Update the x value by subtracting 50 from it, keep the original y value
00414             Sprite.FlipH = true;
00415             position = position with { X = position.X - 25 };
00416         }
00417         Position = position;
00418
00419     }
00420
00421
00425     private void UpdateAnimations() {
00426         if (Input.IsActionJustPressed("light_attack") && !IsDashing && !IsAttacking()) {
00427             if (UseStamina(10)){
00428                 LastAttack = 1;
00429                 AnimationPlayer.Play("light_attack");
00430             }
00431         } else if (Input.IsActionJustPressed("heavy_attack") && !IsDashing && !IsAttacking()) {
00432             if (UseStamina(25)){
00433                 LastAttack = 2;
00434                 AnimationPlayer.Play("heavy_attack");
00435             }
00436         }
00437         if (Input.IsActionPressed("block") && !IsDashing && !IsAttacking() && IsOnFloor()) {
00438             if (UseStamina(0)){
00439                 AnimationPlayer.Play("block");
00440                 LastAttack = 0;
00441             }
00442         }
00443
00444         if (IsOnFloor() && !IsAttacking() && !IsBlocking()) {
00445             LastAttack = 0;
00446             if (Velocity.X == 0) {
00447                 AnimationPlayer.Play("idle");
00448             } else {
00449                 AnimationPlayer.Play("run");
00450             }
00451         } else if (!IsOnFloor() && !IsAttacking() && !IsBlocking()) {
00452             LastAttack = 0;
00453             if (Velocity.Y < 0) {
00454                 AnimationPlayer.Play("jump");
00455             } else if (Velocity.Y > 0) {
00456                 AnimationPlayer.Play("fall");
00457             }
00458         }
00459     }
00460 }
```

5.29 PlayerStats.cs File Reference

Classes

- class PlayerStats

  Klasse für die Spielerstats.

5.30 PlayerStats.cs

Go to the documentation of this file.

```
00001 using System;
00002 using Godot;
00003
00007 public partial class PlayerStats : Node
00008 {
00009
00010     public static PlayerStats Instance { get; private set; }
00011
00012     private String RespawnLevelTag = "intro";
00013     private String CurrentLevelTag = "intro";
00014     private Vector2 SpawnPoint;
00015     private Vector2 Position = new Vector2(-540, 160);
```

```
00016     private int SinAmount;
00017     private float MaxHealthPoints = 100f;
00018     private float CurrentHealth;
00019     private float MaxStamina = 100f;
00020     private float CurrentStamina;
00021     private int BVHealAmount = 25;
00022     private int BVMaxUses = 5;
00023     private int BVCurrentUses;
00024
00025
00029     public override void _Ready(){
00030         CurrentHealth = MaxHealthPoints;
00031         CurrentStamina = MaxStamina;
00032         BVCurrentUses = BVMaxUses;
00033         Instance = this;
00034     }
00035
00040     public String GetRespawnLevelTag() {
00041         return RespawnLevelTag;
00042     }
00043
00048     public void SetRespawnLevelTag(String levelTag) {
00049         RespawnLevelTag = levelTag;
00050     }
00051
00056     public String GetCurrentLevelTag() {
00057         return CurrentLevelTag;
00058     }
00059
00064     public void SetCurrentLevelTag(String levelTag) {
00065         CurrentLevelTag = levelTag;
00066     }
00067
00072     public void SetSpawnPoint(Vector2 spawnPoint) {
00073         SpawnPoint = spawnPoint;
00074     }
00075
00080     public Vector2 GetSpawnPoint(){
00081         return SpawnPoint;
00082     }
00083
00088     public void SetPosition(Vector2 position) {
00089         Position = position;
00090     }
00091
00096     public Vector2 GetPosition(){
00097         return Position;
00098     }
00099
00100
00105     public int GetSinAmount(){
00106         return SinAmount;
00107     }
00108
00113     public void SetSinAmount(int Value) {
00114         // SinAmount muss immer >= 0 sein
00115         SinAmount = Mathf.Max(Value, 0);
00116     }
00117
00122     public float GetMaxHealthPoints(){
00123         return MaxHealthPoints;
00124     }
00125
00130     public void SetMaxHealthPoints(float maxHealthPoints){
00131         // MaxHealthPoints muss immer positiv sein
00132         MaxHealthPoints = Mathf.Max(maxHealthPoints, 1); // Verhindert, dass MaxHealthPoints <= 0 wird
00133     }
00134
00139     public float GetCurrentHealth(){
00140         return CurrentHealth;
00141     }
00142
00147     public void SetCurrentHealth(float Health){
00148         // CurrentHealth darf MaxHealthPoints nicht überschreiten.
00149         CurrentHealth = Mathf.Min(Health, MaxHealthPoints);
00150     }
00151
00156     public void SetMaxStamina(float Value) {
00157         // MaxStamina muss immer positiv sein
00158         MaxStamina = Mathf.Max(Value, 1);
00159     }
00160
00165     public float GetMaxStamina() {
00166         return MaxStamina;
00167     }
00168
00173     public void SetStamina(float Value) {
```

```
00174         // Stellt sicher, dass die CurrentStamina im gültigen Bereich bleibt (zwischen 0 und
      MaxStamina)
00175         CurrentStamina = Mathf.Clamp(Value, 0, MaxStamina);
00176     }
00177
00182     public float GetStamina() {
00183         return CurrentStamina;
00184     }
00185
00190     public void SetBVHealAmount(int Value){
00191         BVHealAmount = Math.Max(0, Value);
00192     }
00193
00198     public int GetBVHealAmount() {
00199         return BVHealAmount;
00200     }
00201
00206     public void SetBVMaxUses(int Value){
00207         BVMaxUses = Math.Max(0, Value);
00208     }
00209
00214     public int GetBVMaxUses() {
00215         return BVMaxUses;
00216     }
00217
00222     public void SetBVCurrentUses(int Value){
00223         BVCurrentUses = Math.Max(0, Value);
00224     }
00225
00230     public int GetBVCurrentUses() {
00231         return BVCurrentUses;
00232     }
00233
00237     public void Reload(){
00238         Instance = new PlayerStats();
00239         Instance._Ready();
00240     }
00241
00242 }
```

5.31 Spike.cs File Reference

Classes

- class Spike

  Klasse für die Spikes.

5.32 Spike.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003
00007 public partial class Spike : Node2D
00008 {
00009     // Variable für Player
00010     private Player Player;
00011
00012
00013     [Export]
00014     private float Damage = 10f;
00015
00020     public override void _Ready()
00021     {
00022         // Zugriff auf Player Node
00023
00024         Player = GetNode<Player>("../../Player");
00025     }
00026
00030     private void OnPlayerBodyEntered(Node body)
00031     {
00032
00033         if (body is Player)
```

```
00034         {
00035             Player = (Player)body; // Instanzvariable setzen
00036             Player.TakeDamage(GetDamage());
00037             Player.SlowPlayer(0.5f);
00038             GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00039             GD.Print("Player entered spike");
00040         }
00041 
00042 
00043     }
00044 
00048     private void OnPlayerBodyExited(Node body)
00049     {
00050         if (body is Player)
00051         {
00052             Player = null; // Instanzvariable zurücksetzen
00053             GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
00054         }
00055     }
00056 
00060     private void OnTimerTimeout()
00061     {
00062         GD.Print("Timer timeout");
00063         Player.TakeDamage(GetDamage());
00064         GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00065     }
00066 
00071     public Damage GetDamage()
00072     {
00073         return new Damage(0, Damage, Vector2.Zero, this);
00074     }
00075 }
```

5.33 SpikeDynamic.cs File Reference

Classes

- class SpikeDynamic

  Klasse für die beweglichen Spikes.

5.34 SpikeDynamic.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003 
00007 public partial class SpikeDynamic : Node2D
00008 {
00009     // Variable für Player
00010     private Player Player;
00011 
00012     [Export]
00013     private float Damage = 10f;
00014 
00020     public override void _Ready()
00021     {
00022         // Zugriff auf Player Node
00023 
00024         Player = GetNode<Player>("../../../Player");
00025     }
00026 
00030     private void OnPlayerBodyEntered(Node body)
00031     {
00032 
00033         if (body is Player)
00034         {
00035             Player = (Player)body; // Instanzvariable setzen
00036             Player.TakeDamage(GetDamage());
00037             Player.SlowPlayer(0.5f);
00038             GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00039             GD.Print("Player entered spike");
00040         }
00041 
```

```
00042
00043     }
00044
00048     private void OnPlayerBodyExited(Node body)
00049     {
00050         if (body is Player)
00051         {
00052             Player = null; // Instanzvariable zurücksetzen
00053             GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
00054         }
00055     }
00056
00060     private void OnTimerTimeout()
00061     {
00062         GD.Print("Timer timeout");
00063         Player.TakeDamage(GetDamage());
00064         GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
00065     }
00066
00071     public Damage GetDamage()
00072     {
00073         return new Damage(0, Damage, Vector2.Zero, this);
00074     }
00075 }
```

5.35 StaminaBar.cs File Reference

Classes

- class StaminaBar

  Klasse für die Ausdauerleiste des Spielers. Synchronisiert die Anzeige der StaminaBar mit der Ausdauer des Spielers.

5.36 StaminaBar.cs

Go to the documentation of this file.

```
00001 using Godot;
00002
00007 public partial class StaminaBar : TextureProgressBar {
00008
00013     public override void _Ready() {
00014         // Setze die maximale Ausdauer der StaminaBar basierend auf der Spieler-MaxStamina
00015         MaxValue = PlayerStats.Instance.GetMaxStamina();
00016         Value = PlayerStats.Instance.GetStamina();
00017     }
00018
00024     public override void _Process(double DeltaTime) {
00025         // Aktualisiere den Wert der StaminaBar basierend auf der aktuellen Ausdauer des Spielers
00026         Value = PlayerStats.Instance.GetStamina();
00027     }
00028 }
```

5.37 StorageManager.cs File Reference

Classes

- class StorageManager

  Klasse für das Speichern und Laden von Daten.

5.38 StorageManager.cs

Go to the documentation of this file.

```
00001 using Godot;
00002 using System;
00003 using System.Collections;
00004
00008 public partial class StorageManager : Node {
00009
00010     public static StorageManager Instance { get; private set; }
00011     private const String PathSettings = "user://settings.txt";
00012     private String[] PathSave = {"user://save1.dat", "user://save2.dat", "user://save3.dat"};
00013     private int LastSaveId = -1;
00014     private int Saves = 0;
00015
00016
00020     public override void _Ready(){
00021         LoadSettings();
00022         Instance = this;
00023     }
00024
00028     public void LoadSettings(){
00029         if(!FileAccess.FileExists(PathSettings)){
00030             return;
00031         }
00032         FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Read);
00033         Saves = (int) File.GetVar();
00034         LastSaveId = (int) File.GetVar();
00035
00036         File.Close();
00037     }
00038
00043     public void LoadGameFile(int id){
00044         if(!FileAccess.FileExists(PathSave[id])){
00045             return;
00046         }
00047         FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Read);
00048         PlayerStats.Instance.SetRespawnLevelTag((String) File.GetVar());
00049         PlayerStats.Instance.SetCurrentLevelTag((String) File.GetVar());
00050         PlayerStats.Instance.SetSpawnPoint((Vector2) File.GetVar());
00051         PlayerStats.Instance.SetPosition((Vector2) File.GetVar());
00052         PlayerStats.Instance.SetSinAmount((int) File.GetVar());
00053         PlayerStats.Instance.SetMaxHealthPoints((float) File.GetVar());
00054         PlayerStats.Instance.SetCurrentHealth((float) File.GetVar());
00055         PlayerStats.Instance.SetMaxStamina((float) File.GetVar());
00056         PlayerStats.Instance.SetStamina((float) File.GetVar());
00057         PlayerStats.Instance.SetBVHealAmount((int) File.GetVar());
00058         PlayerStats.Instance.SetBVMaxUses((int) File.GetVar());
00059         PlayerStats.Instance.SetBVCurrentUses((int) File.GetVar());
00060
00061         File.Close();
00062     }
00063
00068     public void SaveAll(int id){
00069         SaveGameFile(id);
00070         SaveSettings();
00071     }
00072
00076     public void SaveSettings(){
00077         FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Write);
00078         File.StoreVar(Saves);
00079         File.StoreVar(LastSaveId);
00080
00081         File.Close();
00082     }
00083
00088     public void SaveGameFile(int id){
00089         FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Write);
00090         File.StoreVar(PlayerStats.Instance.GetRespawnLevelTag());
00091         File.StoreVar(PlayerStats.Instance.GetCurrentLevelTag());
00092         File.StoreVar(PlayerStats.Instance.GetSpawnPoint());
00093         File.StoreVar(PlayerStats.Instance.GetPosition());
00094         File.StoreVar(PlayerStats.Instance.GetSinAmount());
00095         File.StoreVar(PlayerStats.Instance.GetMaxHealthPoints());
00096         File.StoreVar(PlayerStats.Instance.GetCurrentHealth());
00097         File.StoreVar(PlayerStats.Instance.GetMaxStamina());
00098         File.StoreVar(PlayerStats.Instance.GetStamina());
00099         File.StoreVar(PlayerStats.Instance.GetBVHealAmount());
00100         File.StoreVar(PlayerStats.Instance.GetBVMaxUses());
00101         File.StoreVar(PlayerStats.Instance.GetBVCurrentUses());
00102
00103         File.Close();
00104     }
00105
00110     public void SetLastSaveId(int id){
```

```
00111         LastSaveId = id;
00112     }
00113
00118     public int GetLastSaveId(){
00119         return LastSaveId;
00120     }
00121
00126     public void SetSaves(int Saves){
00127         this.Saves = Saves;
00128     }
00129
00134     public int GetSaves(){
00135         return Saves;
00136     }
00137 }
```

Index