Anfaengerpraktikum

v.0.1.0

Generated by Doxygen 1.12.0

1 anfaengerpraktikum 1

2 Third-Party Notices 3

2.1 License Texts 3

2.1.1 godotengine/godot-cpp 3

2.1.2 libgit2/libgit2 4

2.1.3 libssh2/libssh2 17

2.1.4 OpenSSL 17

3 Namespace Index 21

3.1 Namespace List 21

4 Hierarchical Index 23

4.1 Class Hierarchy 23

5 Class Index 25

5.1 Class List 25

6 File Index 27

6.1 File List 27

7 Namespace Documentation 29

7.1 GdMUT Namespace Reference 29

7.2 GdMUT.Components Namespace Reference 29

8 Class Documentation 31

8.1 BaseEnemy Class Reference 31

8.1.1 Detailed Description 33

8.1.2 Member Enumeration Documentation 33

8.1.2.1 State 33

8.1.3 Member Function Documentation 33

8.1.3.1 _Process() 33

8.1.3.2 _Ready() 34

8.1.3.3 CheckLineOfSight() 34

8.1.3.4 CheckPlayerHit() 35

8.1.3.5 Die() 36

8.1.3.6 FlipRotation() 36

8.1.3.7 HandleMovement() 36

8.1.3.8 IsCloseTo() 37

8.1.3.9 IsDead() 39

8.1.3.10 OnDetectionBodyEntered() 39

8.1.3.11 OnHitboxAreaEntered() 39

8.1.3.12 OnPursuingRadiusBodyExited() 40

8.1.3.13 OnSwordHitBoxBodyEntered() 40

8.1.3.14 Respawn() 41

8.1.3.15 SetRotation() . . . 41
8.1.3.16 TakeDamage() . . . 41
8.1.3.17 UpdateAnimation() . . . 42
8.1.4 Member Data Documentation . . . 42
8.1.4.1 AlreadyHit . . . 42
8.1.4.2 AnimationState . . . 43
8.1.4.3 Armor . . . 43
8.1.4.4 CollisionPolygon . . . 43
8.1.4.5 CurrentHealthPoints . . . 43
8.1.4.6 CurrentStamina . . . 43
8.1.4.7 CurrentTarget . . . 43
8.1.4.8 Damage . . . 44
8.1.4.9 Dead . . . 44
8.1.4.10 FrontCollisionRayCast . . . 44
8.1.4.11 HealthBar . . . 44
8.1.4.12 LeftFallProtection . . . 44
8.1.4.13 LineOfSight . . . 44
8.1.4.14 MainCollision . . . 45
8.1.4.15 MaxHealthPoints . . . 45
8.1.4.16 MaxStamina . . . 45
8.1.4.17 Player . . . 45
8.1.4.18 Pursuing . . . 45
8.1.4.19 Respawnable . . . 45
8.1.4.20 ReturnToStart . . . 46
8.1.4.21 ReturnToStartAfter . . . 46
8.1.4.22 RightFallProtection . . . 46
8.1.4.23 SinAmount . . . 46
8.1.4.24 Speed . . . 46
8.1.4.25 Sprite . . . 46
8.1.4.26 StartPosition . . . 47
8.1.4.27 StartRotation . . . 47
8.1.4.28 SwordHitbox . . . 47
8.1.4.29 TargetPosition . . . 47
8.1.5 Property Documentation . . . 47
8.1.5.1 Id . . . 47
8.2 BloodVial Class Reference . . . 48
8.2.1 Detailed Description . . . 48
8.2.2 Member Function Documentation . . . 48
8.2.2.1 _Ready() . . . 48
8.2.2.2 AddMaxUses() . . . 48
8.2.2.3 LevelHealAmount() . . . 49
8.2.2.4 ResetUses() . . . 49

8.2.2.5 UseBloodVial() 49
8.3 Boss1 Class Reference 50
8.3.1 Detailed Description 52
8.3.2 Member Function Documentation 52
8.3.2.1 _Process() 52
8.3.2.2 _Ready() 52
8.3.2.3 HandleRegeneration() 52
8.3.2.4 ReviveEnemies() 53
8.3.2.5 ShowPopupMessage() 53
8.3.2.6 StartGlowing() 54
8.3.3 Member Data Documentation 54
8.3.3.1 EnemiesRevived 54
8.3.3.2 RegenAmount 55
8.3.3.3 RegenCooldown 55
8.3.3.4 RegenTimer 55
8.4 Checkpoint Class Reference 55
8.4.1 Detailed Description 56
8.4.2 Member Function Documentation 56
8.4.2.1 _Ready() 56
8.4.2.2 OnPlayerBodyEntered() 56
8.4.3 Member Data Documentation 56
8.4.3.1 Player 56
8.5 Damage Class Reference 57
8.5.1 Detailed Description 57
8.5.2 Constructor & Destructor Documentation 57
8.5.2.1 Damage() 57
8.5.3 Member Function Documentation 58
8.5.3.1 GetPhysicalDMG() 58
8.5.3.2 GetPushAmount() 58
8.5.3.3 GetSource() 58
8.5.3.4 GetTrueDMG() 59
8.5.4 Member Data Documentation 59
8.5.4.1 PhysicalDMG 59
8.5.4.2 PushAmount 59
8.5.4.3 Source 59
8.5.4.4 TrueDMG 60
8.6 Door Class Reference 60
8.6.1 Detailed Description 61
8.6.2 Member Function Documentation 61
8.6.2.1 _Ready() 61
8.6.2.2 OnPlayerBodyEntered() 61
8.6.3 Member Data Documentation 61

8.6.3.1 Spawn 61
8.6.4 Property Documentation 62
8.6.4.1 DestinationDoorTag 62
8.6.4.2 DestinationLevelTag 62
8.6.4.3 SpawnDirection 62
8.7 HealthBar Class Reference 62
8.7.1 Detailed Description 63
8.7.2 Member Function Documentation 63
8.7.2.1 _Process() 63
8.7.2.2 _Ready() 63
8.8 Hud Class Reference 64
8.8.1 Detailed Description 64
8.8.2 Member Function Documentation 64
8.8.2.1 _Process() 64
8.8.2.2 _Ready() 65
8.8.2.3 OnResumeButtonPressed() 65
8.8.2.4 OnSaveButtonPressed() 65
8.8.2.5 OnSaveMenuButtonPressed() 66
8.8.2.6 OnSaveQuitButtonPressed() 66
8.8.2.7 TogglePause() 66
8.8.3 Member Data Documentation 66
8.8.3.1 AnimationPlayer 66
8.8.3.2 Buttons 67
8.8.3.3 Enabled 67
8.9 Interactable Class Reference 67
8.9.1 Detailed Description 68
8.9.2 Member Function Documentation 68
8.9.2.1 _Process() 68
8.9.2.2 _Ready() 68
8.9.2.3 OnAreaBodyExited() 68
8.9.3 Member Data Documentation 69
8.9.3.1 Area 69
8.9.3.2 Player 69
8.9.3.3 PopUp 69
8.9.3.4 TextLabel 69
8.9.4 Property Documentation 70
8.9.4.1 Text 70
8.10 LevelManager Class Reference 70
8.10.1 Detailed Description 70
8.10.2 Member Function Documentation 71
8.10.2.1 _Ready() 71
8.10.2.2 OnLevelSpawn() 71

8.11 MainMenu Class Reference . . . 72
8.11.1 Detailed Description . . . 73
8.11.2 Member Function Documentation . . . 73
8.11.2.1 _Ready() . . . 73
8.11.2.2 Change() . . . 74
8.11.2.3 OnBackButtonPressed() . . . 74
8.11.2.4 OnContinueButtonPressed() . . . 75
8.11.2.5 OnDeleteConfirmationCanceled() . . . 75
8.11.2.6 OnDeleteConfirmationCloseRequested() . . . 75
8.11.2.7 OnDeleteConfirmationConfirmed() . . . 75
8.11.2.8 OnLoadGameButtonPressed() . . . 76
8.11.2.9 OnNewGameButtonPressed() . . . 76
8.11.2.10 OnQuitButtonPressed() . . . 76
8.11.2.11 OnSave1DeletePressed() . . . 76
8.11.2.12 OnSave1SelectPressed() . . . 77
8.11.2.13 OnSave2DeletePressed() . . . 77
8.11.2.14 OnSave2SelectPressed() . . . 77
8.11.2.15 OnSave3DeletePressed() . . . 78
8.11.2.16 OnSave3SelectPressed() . . . 78
8.11.3 Member Data Documentation . . . 78
8.11.3.1 ContinueButton . . . 78
8.11.3.2 DeleteButton . . . 78
8.11.3.3 DeleteConfirmation . . . 79
8.11.3.4 InfoLabel . . . 79
8.11.3.5 MenuState . . . 79
8.11.3.6 Navigation . . . 79
8.11.3.7 SaveLabel . . . 79
8.11.3.8 SavesContainer . . . 79
8.11.3.9 SaveToDelete . . . 80
8.11.3.10 SelectButton . . . 80
8.12 MainMenuBackground Class Reference . . . 80
8.12.1 Detailed Description . . . 80
8.12.2 Member Function Documentation . . . 80
8.12.2.1 _Process() . . . 80
8.12.3 Member Data Documentation . . . 81
8.12.3.1 ScrollSpeed . . . 81
8.13 NavigationManager Class Reference . . . 81
8.13.1 Detailed Description . . . 82
8.13.2 Member Function Documentation . . . 82
8.13.2.1 _Ready() . . . 82
8.13.2.2 DeferredChangeScene() . . . 82
8.13.2.3 GoToLevel() . . . 83

8.13.2.4 OnTriggerPlayerSpawnEventHandler() . . . 84
8.13.2.5 TriggerPlayerSpawn() . . . 84
8.13.3 Member Data Documentation . . . 84
8.13.3.1 SceneBoss . . . 84
8.13.3.2 SceneIntro . . . 84
8.13.3.3 SceneLevel1 . . . 85
8.13.3.4 SceneLevelOne . . . 85
8.13.3.5 SceneLevelTwo . . . 85
8.13.3.6 SceneMainMenu . . . 85
8.13.4 Property Documentation . . . 85
8.13.4.1 Instance . . . 85
8.13.4.2 SpawnDoorTag . . . 86
8.14 Player Class Reference . . . 86
8.14.1 Detailed Description . . . 88
8.14.2 Member Function Documentation . . . 88
8.14.2.1 _PhysicsProcess() . . . 88
8.14.2.2 _Ready() . . . 88
8.14.2.3 CreateDashEffect() . . . 89
8.14.2.4 DashInProgress() . . . 89
8.14.2.5 GetBloodVials() . . . 90
8.14.2.6 GetDamage() . . . 90
8.14.2.7 HandleJump() . . . 91
8.14.2.8 HandleMovement() . . . 91
8.14.2.9 IsAttacking() . . . 92
8.14.2.10 IsBlocking() . . . 92
8.14.2.11 MaxHeal() . . . 93
8.14.2.12 OnSpawn() . . . 93
8.14.2.13 RegenerateStamina() . . . 93
8.14.2.14 Respawn() . . . 94
8.14.2.15 SetSinAmount() . . . 94
8.14.2.16 SlowPlayer() . . . 94
8.14.2.17 StartDash() . . . 95
8.14.2.18 StopDash() . . . 95
8.14.2.19 TakeDamage() . . . 95
8.14.2.20 UpdateAnimations() . . . 96
8.14.2.21 UseStamina() . . . 97
8.14.3 Member Data Documentation . . . 97
8.14.3.1 AnimationPlayer . . . 97
8.14.3.2 BloodVials . . . 97
8.14.3.3 CanDash . . . 98
8.14.3.4 DashDirection . . . 98
8.14.3.5 DashEffect . . . 98

8.14.3.6 DashSpeed 98
8.14.3.7 DashTimer 98
8.14.3.8 DashTrailInterval 98
8.14.3.9 DashTrailTimer 99
8.14.3.10 HauptHitbox 99
8.14.3.11 IsDashing 99
8.14.3.12 JUMP_VELOCITY 99
8.14.3.13 JumpCount 99
8.14.3.14 JumpMax 99
8.14.3.15 LastAttack 100
8.14.3.16 PlayerHitbox 100
8.14.3.17 SinDisplay 100
8.14.3.18 SPEED 100
8.14.3.19 Sprite 100
8.14.3.20 SwordCollision 100
8.14.3.21 TimeSinceLastStaminaUse 101
8.15 PlayerStats Class Reference 101
8.15.1 Detailed Description 103
8.15.2 Member Function Documentation 103
8.15.2.1 _Ready() 103
8.15.2.2 GetBVCurrentUses() 103
8.15.2.3 GetBVHealAmount() 103
8.15.2.4 GetBVMaxUses() 104
8.15.2.5 GetCurrentHealth() 104
8.15.2.6 GetCurrentLevelTag() 104
8.15.2.7 GetMaxHealthPoints() 105
8.15.2.8 GetMaxStamina() 105
8.15.2.9 GetPosition() 105
8.15.2.10 GetRespawnLevelTag() 106
8.15.2.11 GetSinAmount() 106
8.15.2.12 GetSpawnPoint() 106
8.15.2.13 GetStamina() 107
8.15.2.14 Reload() 107
8.15.2.15 SetBVCurrentUses() 107
8.15.2.16 SetBVHealAmount() 107
8.15.2.17 SetBVMaxUses() 108
8.15.2.18 SetCurrentHealth() 108
8.15.2.19 SetCurrentLevelTag() 108
8.15.2.20 SetMaxHealthPoints() 109
8.15.2.21 SetMaxStamina() 109
8.15.2.22 SetPosition() 109
8.15.2.23 SetRespawnLevelTag() 110

8.15.2.24 SetSinAmount() 110
8.15.2.25 SetSpawnPoint() 110
8.15.2.26 SetStamina() 111
8.15.3 Member Data Documentation 111
8.15.3.1 BVCurrentUses 111
8.15.3.2 BVHealAmount 111
8.15.3.3 BVMaxUses 112
8.15.3.4 CurrentHealth 112
8.15.3.5 CurrentLevelTag 112
8.15.3.6 CurrentStamina 112
8.15.3.7 MaxHealthPoints 112
8.15.3.8 MaxStamina 112
8.15.3.9 Position 113
8.15.3.10 RespawnLevelTag 113
8.15.3.11 SinAmount 113
8.15.3.12 SpawnPoint 113
8.15.4 Property Documentation 113
8.15.4.1 Instance 113
8.16 Spike Class Reference 114
8.16.1 Detailed Description 114
8.16.2 Member Function Documentation 114
8.16.2.1 _Ready() 114
8.16.2.2 GetDamage() 115
8.16.2.3 OnPlayerBodyEntered() 115
8.16.2.4 OnPlayerBodyExited() 115
8.16.2.5 OnTimerTimeout() 116
8.16.3 Member Data Documentation 116
8.16.3.1 Damage 116
8.16.3.2 Player 116
8.17 SpikeDynamic Class Reference 116
8.17.1 Detailed Description 117
8.17.2 Member Function Documentation 117
8.17.2.1 _Ready() 117
8.17.2.2 GetDamage() 118
8.17.2.3 OnPlayerBodyEntered() 118
8.17.2.4 OnPlayerBodyExited() 118
8.17.2.5 OnTimerTimeout() 119
8.17.3 Member Data Documentation 119
8.17.3.1 Damage 119
8.17.3.2 Player 119
8.18 StaminaBar Class Reference 119
8.18.1 Detailed Description 120

8.18.2 Member Function Documentation 120
8.18.2.1 _Process() 120
8.18.2.2 _Ready() 120
8.19 StorageManager Class Reference 121
8.19.1 Detailed Description 122
8.19.2 Member Function Documentation 122
8.19.2.1 _Ready() 122
8.19.2.2 GetLastSaveId() 122
8.19.2.3 GetSaves() 122
8.19.2.4 LoadGameFile() 122
8.19.2.5 LoadSettings() 123
8.19.2.6 SaveAll() 123
8.19.2.7 SaveGameFile() 124
8.19.2.8 SaveSettings() 124
8.19.2.9 SetLastSaveId() 125
8.19.2.10 SetSaves() 125
8.19.3 Member Data Documentation 125
8.19.3.1 LastSaveId 125
8.19.3.2 PathSave 126
8.19.3.3 PathSettings 126
8.19.3.4 Saves 126
8.19.4 Property Documentation 126
8.19.4.1 Instance 126

9 File Documentation **127**
9.1 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/Dock.cs File Reference 127
9.2 Dock.cs 127
9.3 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/GDMUT.cs File Reference 129
9.4 GDMUT.cs 129
9.5 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/MethodResult.cs File Reference 130
9.6 MethodResult.cs 130
9.7 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/Result.cs File Reference 131
9.8 Result.cs 131
9.9 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestFunction.cs File Reference 131
9.10 TestFunction.cs 131
9.11 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestLoader.cs File Reference 131
9.12 TestLoader.cs 131

9.13 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestResult.cs File Reference 133
9.14 TestResult.cs 133
9.15 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/godot-git-plugin/THIRDPARTY.md File Reference 134
9.16 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/↩
README.md File Reference 134
9.17 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
BaseEnemy.cs File Reference 134
9.18 BaseEnemy.cs 134
9.19 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
BloodVial.cs File Reference 138
9.20 BloodVial.cs 138
9.21 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Boss1.cs File Reference 139
9.22 Boss1.cs 139
9.23 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Checkpoint.cs File Reference 140
9.24 Checkpoint.cs 140
9.25 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Damage.cs File Reference 141
9.26 Damage.cs 141
9.27 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Door.cs File Reference 141
9.28 Door.cs 142
9.29 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
HealthBar.cs File Reference 142
9.30 HealthBar.cs 142
9.31 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Hud.cs File Reference 143
9.32 Hud.cs 143
9.33 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Interactable.cs File Reference 143
9.34 Interactable.cs 144
9.35 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
LevelManager.cs File Reference 144
9.36 LevelManager.cs 144
9.37 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
MainMenu.cs File Reference 145
9.38 MainMenu.cs 145
9.39 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
MainMenuBackground.cs File Reference 147
9.40 MainMenuBackground.cs 147
9.41 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
NavigationManager.cs File Reference 148
9.42 NavigationManager.cs 148

9.43 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Player.cs File Reference 149
9.44 Player.cs 149
9.45 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
PlayerStats.cs File Reference 153
9.46 PlayerStats.cs 154
9.47 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
Spike.cs File Reference 155
9.48 Spike.cs 155
9.49 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
SpikeDynamic.cs File Reference 156
9.50 SpikeDynamic.cs 156
9.51 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
StaminaBar.cs File Reference 157
9.52 StaminaBar.cs 157
9.53 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/↩
StorageManager.cs File Reference 158
9.54 StorageManager.cs 158
9.55 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/Test↩
Class.cs File Reference 159
9.56 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/Test↩
Class.cs 159

Index **161**

Chapter 1

anfaengerpraktikum

Praktikum agile Computerspielentwicklung.

Chapter 2

Third-Party Notices

The Godot Git Plugin source code uses the following third-party source code:

1. godotengine/godot-cpp - MIT License - https://github.com/godotengine/godot-cpp/tree/02336831735
2. libgit2/libgit2 - GPLv2 with a special Linking Exception - https://github.com/libgit2/libgit2/tree/b7bad5
3. libssh2/libssh2 - BSD-3-Clause License - https://github.com/libssh2/libssh2/tree/635caa90787220ac

We also link to these third-party libraries (only in the compiled binary form):

1. OpenSSL - Only on Linux and MacOS - OpenSSL License - http://www.openssl.↩ org/source/openssl-1.1.1s.tar.gz

2.1 License Texts

2.1.1 godotengine/godot-cpp

2.1.2 libgit2/libgit2

```
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
```

```
and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

  You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

  2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application.  Therefore, Subsection 2d requires that any
    application-supplied function or table used by this function must
    be optional: if the application does not supply it, the square
    root function must still compute square roots.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Library.

In addition, mere aggregation of another work not based on the Library
with the Library (or with a work based on the Library) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public
License instead of this License to a given copy of the Library.  To do
this, you must alter all the notices that refer to this License, so
that they refer to the ordinary GNU General Public License, version 2,
instead of to this License.  (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify
that version instead if you wish.)  Do not make any other change in
these notices.

  Once this change is made in a given copy, it is irreversible for
that copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of
the Library into a program that is not a library.

  4. You may copy and distribute the Library (or a portion or
derivative of it, under Section 2) in object code or executable form
under the terms of Sections 1 and 2 above provided that you accompany
it with the complete corresponding machine-readable source code, which
must be distributed under the terms of Sections 1 and 2 above on a
medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy
```

```
from a designated place, then offering equivalent access to copy the
source code from the same place satisfies the requirement to
distribute the source code, even though third parties are not
compelled to copy the source along with the object code.

  5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being compiled or
linked with it, is called a "work that uses the Library".  Such a
work, in isolation, is not a derivative work of the Library, and
therefore falls outside the scope of this License.

  However, linking a "work that uses the Library" with the Library
creates an executable that is a derivative of the Library (because it
contains portions of the Library), rather than a "work that uses the
library".  The executable is therefore covered by this License.
Section 6 states terms for distribution of such executables.

  When a "work that uses the Library" uses material from a header file
that is part of the Library, the object code for the work may be a
derivative work of the Library even though the source code is not.
Whether this is true is especially significant if the work can be
linked without the Library, or if the work is itself a library.  The
threshold for this to be true is not precisely defined by law.

  If such an object file uses only numerical parameters, data
structure layouts and accessors, and small macros and small inline
functions (ten lines or less in length), then the use of the object
file is unrestricted, regardless of whether it is legally a derivative
work.  (Executables containing this object code plus portions of the
Library will still fall under Section 6.)

  Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6,
whether or not they are linked directly with the Library itself.

  6. As an exception to the Sections above, you may also combine or
link a "work that uses the Library" with the Library to produce a
work containing portions of the Library, and distribute that work
under terms of your choice, provided that the terms permit
modification of the work for the customer’s own use and reverse
engineering for debugging such modifications.

  You must give prominent notice with each copy of the work that the
Library is used in it and that the Library and its use are covered by
this License.  You must supply a copy of this License.  If the work
during execution displays copyright notices, you must include the
copyright notice for the Library among them, as well as a reference
directing the user to the copy of this License.  Also, you must do one
of these things:

    a) Accompany the work with the complete corresponding
    machine-readable source code for the Library including whatever
    changes were used in the work (which must be distributed under
    Sections 1 and 2 above); and, if the work is an executable linked
    with the Library, with the complete machine-readable "work that
    uses the Library", as object code and/or source code, so that the
    user can modify the Library and then relink to produce a modified
    executable containing the modified Library.  (It is understood
    that the user who changes the contents of definitions files in the
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

    b) Use a suitable shared library mechanism for linking with the
    Library.  A suitable mechanism is one that (1) uses at run time a
    copy of the library already present on the user’s computer system,
    rather than copying library functions into the executable, and (2)
    will operate properly with a modified version of the library, if
    the user installs one, as long as the modified version is
    interface-compatible with the version that the work was made with.

    c) Accompany the work with a written offer, valid for at
    least three years, to give the same user the materials
    specified in Subsection 6a, above, for a charge no more
    than the cost of performing this distribution.

    d) If distribution of the work is made by offering access to copy
    from a designated place, offer equivalent access to copy the above
    specified materials from the same place.

    e) Verify that the user has already received a copy of these
    materials or that you have already sent this user a copy.

  For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it.  However, as a special exception,
the materials to be distributed need not include anything that is
```

2.1.3 libssh2/libssh2

2.1.4 OpenSSL

```
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

GdMUT 29
GdMUT.Components 29

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

- AnimatedSprite2D
  - Interactable . . . 67
- Area2D
  - Door . . . 60
- CanvasLayer
  - Hud . . . 64
- CharacterBody2D
  - BaseEnemy . . . 31
    - Boss1 . . . 50
  - Player . . . 86
- Damage . . . 57
- Label
  - BloodVial . . . 48
- Node
  - NavigationManager . . . 81
  - PlayerStats . . . 101
  - StorageManager . . . 121
- Node2D
  - Checkpoint . . . 55
  - LevelManager . . . 70
  - MainMenu . . . 72
  - Spike . . . 114
  - SpikeDynamic . . . 116
- ParallaxLayer
  - MainMenuBackground . . . 80
- TextureProgressBar
  - HealthBar . . . 62
  - StaminaBar . . . 119

Chapter 5

Class Index

5.1 Class List

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

BaseEnemy
Klasse für einen einfachen Gegner 31
BloodVial
Klasse für die Interaktion zum heilen 48
Boss1
Klasse für einen stärkeren Boss-Gegner, der von BaseEnemy erbt 50
Checkpoint 55
Damage
Repräsentiert den Schaden, der von Charakteren oder Gegnern verursacht wird. Beinhaltet physischen Schaden, wahren Schaden und den Rückstoßeffekt 57
Door
Klasse für die Tür 60
HealthBar
Klasse für die Gesundheitsleiste des Spielers. Synchronisiert die Anzeige der HealthBar mit den Lebenspunkten des Spielers 62
Hud
Klasse für das PauseMenu 64
Interactable
Klasse für Interaktion 67
LevelManager
Klasse für den LevelManager Diese Klasse verwaltet den Levelwechsel und die Spielerpositionierung 70
MainMenu
Klasse für das MainMenu 72
MainMenuBackground
Klasse für die MainMenuBackground-Animation 80
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 81
Player
Klasse für den Spielercharakter. Verwaltet Bewegung, Sprünge, Angriffe und Animationen . . . 86
PlayerStats
Klasse für die Spielerstats 101
Spike
Klasse für die Spikes 114

SpikeDynamic
Klasse für die beweglichen Spikes 116
StaminaBar
Klasse für die Ausdauerleiste des Spielers. Synchronisiert die Anzeige der StaminaBar mit der Ausdauer des Spielers 119
StorageManager
Klasse für das Speichern und Laden von Daten 121

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/TestClass.cs 159
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/Dock.cs 127
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/GDMUT.cs 129
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/MethodResult.cs 130
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/Result.cs 131
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestFunction.cs 131
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestLoader.cs 131
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/↩
GDMUT/TestResult.cs 133
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BaseEnemy.cs
134
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BloodVial.cs
138
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Boss1.cs
139
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Checkpoint.cs
140
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Damage.cs
141
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Door.cs
141
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/HealthBar.cs
142
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Hud.cs
143
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Interactable.cs
143
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/LevelManager.cs
144

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/MainMenu.cs
145
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/MainMenuBackground.cs
147
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/NavigationManager.cs
148
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Player.cs
149
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/PlayerStats.cs
153
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Spike.cs
155
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/SpikeDynamic.cs
156
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/StaminaBar.cs
157
C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/StorageManager.cs
158

Chapter 7

Namespace Documentation

7.1 GdMUT Namespace Reference

Namespaces

- namespace Components

Classes

- class **TestClass**

  This is a test class for GDMUT. This is purely for demonstration. If you added this into your project, feel free to delete it =).

7.2 GdMUT.Components Namespace Reference

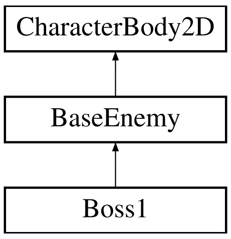
Chapter 8

Class Documentation

8.1 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.
- bool IsDead ()

  Gibt boolean Dead zurück.
- void Respawn ()

  Wird aufgerufen wenn der Gegner respawnt.

Protected Member Functions

- virtual void UpdateAnimation ()

  Aktualisiert die Animationen des Gegners.

Protected Attributes

- float Damage = 20f
- bool Dead = false
- bool Respawnable = true
- float MaxHealthPoints = 100f
- float Armor = 20f
- float MaxStamina = 1f
- float Speed = 10
- int SinAmount = 10
- double ReturnToStartAfter = 5
- float CurrentHealthPoints
- float CurrentStamina
- double ReturnToStart
- bool Pursuing = false
- Node2D CurrentTarget = null
- Vector2 TargetPosition = Vector2.Inf
- Vector2 StartPosition
- bool StartRotation = false
- bool AlreadyHit = false
- AnimatedSprite2D Sprite
- CollisionPolygon2D CollisionPolygon
- Area2D SwordHitbox
- CollisionShape2D MainCollision
- 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 TakeDamage (Damage DMG)

  Verarbeitet zugefügten Schaden.
- void CheckPlayerHit ()

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

  Wird aufgerufen wenn der Gegner stirbt.
- 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

8.1.1 Detailed Description

Klasse für einen einfachen Gegner.

Definition at line 7 of file BaseEnemy.cs.

8.1.2 Member Enumeration Documentation

8.1.2.1 State

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

Enumerator

IDLE	
WALK	
ATTACK	
TAKE_HIT	

Definition at line 10 of file BaseEnemy.cs.

```
                        {
            IDLE, WALK, ATTACK, TAKE_HIT
        }
```

8.1.3 Member Function Documentation

8.1.3.1 _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.

```
    {
        HandleMovement(DeltaTime);
        if(CurrentStamina < MaxStamina){
            CurrentStamina += (float) DeltaTime;
            Velocity = Velocity * 0.8f;
        }
        if (!IsOnFloor() && !Dead) {
            Velocity += GetGravity() * (float)DeltaTime;
        }
        UpdateAnimation();
        MoveAndSlide();
        CheckPlayerHit();
    }
```

References CheckPlayerHit(), CurrentStamina, Dead, HandleMovement(), MaxStamina, and UpdateAnimation().

8.1.3.2 _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.

```
    {
        Sprite = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
        CollisionPolygon = GetNode<CollisionPolygon2D>("detection/CollisionPolygon2D");
        SwordHitbox = GetNode<Area2D>("AnimatedSprite2D/SwordHitBox");
        MainCollision = GetNode<CollisionShape2D>("MainCollision");
        FrontCollisionRayCast = GetNode<RayCast2D>("FrontCollisionRayCast");
        LineOfSight = GetNode<RayCast2D>("LineOfSight");
        LeftFallProtection = GetNode<RayCast2D>("LeftFallProtection");
        RightFallProtection = GetNode<RayCast2D>("RightFallProtection");
        HealthBar = GetNode<TextureProgressBar>("HealthBar");
        Player = GetNode<Player>("../../Player");

        CurrentHealthPoints = MaxHealthPoints;
        CurrentStamina = MaxStamina;
        ReturnToStart = ReturnToStartAfter;
        StartPosition = Position;
        StartRotation = Sprite.FlipH;

        HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
    }
```

References CollisionPolygon, CurrentHealthPoints, CurrentStamina, FrontCollisionRayCast, LeftFallProtection, LineOfSight, MainCollision, MaxHealthPoints, MaxStamina, ReturnToStart, ReturnToStartAfter, RightFallProtection, Sprite, StartPosition, StartRotation, and SwordHitbox.

8.1.3.3 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 334 of file BaseEnemy.cs.

```
                                        {
        Vector2 offset = Vector2.Zero;
        offset.Y = -14;
        LineOfSight.TargetPosition = body.Position + offset - (Position + LineOfSight.Position);
        if(LineOfSight.IsColliding()){
            return LineOfSight.GetCollider() == body;
        }
        return true;
    }
```

References LineOfSight.

Referenced by OnDetectionBodyEntered().

8.1.3.4 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.

```
                             {
        if(Dead) return;
        if(Sprite.Animation != "attack"){
            AlreadyHit = false;
            if(Sprite.Animation == "take_hit" || CurrentStamina < MaxStamina) return;
            Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    Sprite.Play("attack");
                }
            }
            return;
        }
        if(AlreadyHit) return;
        if(Sprite.Frame >= 6){
            CurrentStamina = 0;
            Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    Player.TakeDamage(new Damage(Damage, 0f, Vector2.Zero, this));
                    AlreadyHit = true;
                    break;
                }
            }
        }

    }
```

References AlreadyHit, CurrentStamina, Damage, Dead, MaxStamina, Sprite, SwordHitbox, and Player.TakeDamage().

Referenced by _Process().

8.1.3.5 Die()

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

Wird aufgerufen wenn der Gegner stirbt.

Definition at line 305 of file BaseEnemy.cs.

```
                        {
        Dead = true;
        Velocity = Vector2.Zero;
        MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, true);

        Sprite.Play("death");
        HealthBar.SetVisible(false);
        Player.SetSinAmount(PlayerStats.Instance.GetSinAmount() + SinAmount);

    }
```

References Dead, PlayerStats.GetSinAmount(), PlayerStats.Instance, MainCollision, Player.SetSinAmount(), SinAmount, and Sprite.

Referenced by TakeDamage().

8.1.3.6 FlipRotation()

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

Spiegelt die Orientierung aller zu dem Gegner gehörender Nodes.

Definition at line 347 of file BaseEnemy.cs.

```
                                 {
        Sprite.FlipH = !Sprite.FlipH;
        CollisionPolygon.RotationDegrees = Math.Abs(CollisionPolygon.RotationDegrees -180);
        SwordHitbox.RotationDegrees = Math.Abs(SwordHitbox.RotationDegrees -180);
        FrontCollisionRayCast.RotationDegrees = Math.Abs(FrontCollisionRayCast.RotationDegrees - 180);
    }
```

References CollisionPolygon, FrontCollisionRayCast, Sprite, and SwordHitbox.

Referenced by HandleMovement().

8.1.3.7 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     }
```

References AnimationState, CurrentTarget, Dead, FlipRotation(), FrontCollisionRayCast, IsCloseTo(), LeftFallProtection, Pursuing, ReturnToStart, ReturnToStartAfter, RightFallProtection, SetRotation(), Speed, Sprite, StartPosition, StartRotation, and TargetPosition.

Referenced by _Process().

8.1.3.8 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 378 of file BaseEnemy.cs.

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

Referenced by HandleMovement().

8.1.3.9 IsDead()

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

Gibt boolean Dead zurück.

Returns

bool ob Gegner tot ist.

Definition at line 266 of file BaseEnemy.cs.

```
                            {
        return Dead;
    }
```

References Dead.

Referenced by Boss1.ReviveEnemies().

8.1.3.10 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.

```
                                             {
        if(CheckLineOfSight(body)){
            Pursuing = true;
            CurrentTarget = body;
        }
    }
```

References CheckLineOfSight(), CurrentTarget, and Pursuing.

8.1.3.11 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.

```
                                                {
        Player Player1 = (Player) area.GetParent().GetParent();
        TakeDamage(Player1.GetDamage());
    }
```

References Player.GetDamage(), Player, and TakeDamage().

8.1.3.12 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.

```
                                                       {
        if(body == CurrentTarget){
            Pursuing = false;
            CurrentTarget = null;
        }
    }
```

References CurrentTarget, and Pursuing.

8.1.3.13 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.

```
                                                     {
        if(Dead) return;
        Sprite.Play("attack");
    }
```

References Dead, and Sprite.

8.1.3.14 Respawn()

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

Wird aufgerufen wenn der Gegner respawnt.

Definition at line 319 of file BaseEnemy.cs.

```
    {
        Dead = false;
        CurrentHealthPoints = MaxHealthPoints;
        HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
        MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, false);
        HealthBar.SetVisible(true);
        Sprite.Play("idle");
    }
```

References CurrentHealthPoints, Dead, MainCollision, MaxHealthPoints, and Sprite.

Referenced by Boss1.ReviveEnemies().

8.1.3.15 SetRotation()

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

Setzt Orientierung aller zu dem Gegner gehörender Nodes.

Parameters

Rotation	Die neue Orientierung.

Definition at line 358 of file BaseEnemy.cs.

```
                                        {
        Sprite.FlipH = Rotation ^ StartRotation; // XOR mit StartRotation
        if(Rotation){
            CollisionPolygon.RotationDegrees = 180;
            SwordHitbox.RotationDegrees = 180;
            FrontCollisionRayCast.RotationDegrees = 180;
        } else {
            CollisionPolygon.RotationDegrees = 0;
            SwordHitbox.RotationDegrees = 0;
            FrontCollisionRayCast.RotationDegrees = 0;
        }
    }
```

References StartRotation.

Referenced by HandleMovement().

8.1.3.16 TakeDamage()

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

Verarbeitet zugefügten Schaden.

Parameters

DMG	Schaden der zugefügt wird.

Definition at line 245 of file BaseEnemy.cs.

```
                                      {
        if(Dead) {
            return;
        }
        CurrentHealthPoints -= DMG.GetPhysicalDMG() * (1 - Armor / 100.0f) + DMG.GetTrueDMG();
        Position += DMG.GetPushAmount();
        if(CurrentHealthPoints <= 0){
            Die();
        } else {
            Sprite.Play("take_hit");
            if(DMG.GetSource() == Player){
                Pursuing = true;
                CurrentTarget = Player;
            }
        }
    }
```

References Armor, CurrentHealthPoints, CurrentTarget, Dead, Die(), Damage.GetPhysicalDMG(), Damage.GetPushAmount(), Damage.GetSource(), Damage.GetTrueDMG(), Player, Pursuing, and Sprite.

Referenced by OnHitboxAreaEntered().

8.1.3.17 UpdateAnimation()

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

Aktualisiert die Animationen des Gegners.

Definition at line 216 of file BaseEnemy.cs.

```
                                           {
        if(Dead) return;
        if(!((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying())){
            switch(AnimationState){
                case State.IDLE:
                    Sprite.Play("idle");
                    break;

                case State.WALK:
                    Sprite.Play("walk");
                    break;

                case State.ATTACK:
                    Sprite.Play("attack");
                    break;

                case State.TAKE_HIT:
                    Sprite.Play("take_hit");
                    break;
            }
        }
        HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;

    }
```

References AnimationState, CurrentHealthPoints, Dead, MaxHealthPoints, and Sprite.

Referenced by _Process().

8.1.4 Member Data Documentation

8.1.4.1 AlreadyHit

`bool BaseEnemy.AlreadyHit = false [protected]`

Definition at line 46 of file BaseEnemy.cs.

Referenced by CheckPlayerHit().

8.1.4.2 AnimationState

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

Definition at line 45 of file BaseEnemy.cs.

Referenced by HandleMovement(), and UpdateAnimation().

8.1.4.3 Armor

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

Definition at line 24 of file BaseEnemy.cs.

Referenced by Boss1._Process(), Boss1._Ready(), and TakeDamage().

8.1.4.4 CollisionPolygon

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

Definition at line 50 of file BaseEnemy.cs.

Referenced by _Ready(), and FlipRotation().

8.1.4.5 CurrentHealthPoints

```
float BaseEnemy.CurrentHealthPoints  [protected]
```

Definition at line 37 of file BaseEnemy.cs.

Referenced by Boss1._Process(), _Ready(), Boss1._Ready(), Boss1.HandleRegeneration(), Respawn(), TakeDamage(), and UpdateAnimation().

8.1.4.6 CurrentStamina

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

Definition at line 38 of file BaseEnemy.cs.

Referenced by _Process(), _Ready(), and CheckPlayerHit().

8.1.4.7 CurrentTarget

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

Definition at line 41 of file BaseEnemy.cs.

Referenced by HandleMovement(), OnDetectionBodyEntered(), OnPursuingRadiusBodyExited(), and TakeDamage().

8.1.4.8 Damage

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

Definition at line 16 of file BaseEnemy.cs.

Referenced by CheckPlayerHit().

8.1.4.9 Dead

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

Definition at line 18 of file BaseEnemy.cs.

Referenced by _Process(), CheckPlayerHit(), Die(), HandleMovement(), IsDead(), OnSwordHitBoxBodyEntered(), Respawn(), TakeDamage(), and UpdateAnimation().

8.1.4.10 FrontCollisionRayCast

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

Definition at line 53 of file BaseEnemy.cs.

Referenced by _Ready(), FlipRotation(), and HandleMovement().

8.1.4.11 HealthBar

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

Definition at line 57 of file BaseEnemy.cs.

8.1.4.12 LeftFallProtection

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

Definition at line 55 of file BaseEnemy.cs.

Referenced by _Ready(), and HandleMovement().

8.1.4.13 LineOfSight

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

Definition at line 54 of file BaseEnemy.cs.

Referenced by _Ready(), and CheckLineOfSight().

8.1.4.14 MainCollision

```
CollisionShape2D BaseEnemy.MainCollision  [protected]
```

Definition at line 52 of file BaseEnemy.cs.

Referenced by _Ready(), Die(), and Respawn().

8.1.4.15 MaxHealthPoints

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

Definition at line 22 of file BaseEnemy.cs.

Referenced by Boss1._Process(), _Ready(), Boss1._Ready(), Boss1.HandleRegeneration(), Respawn(), and UpdateAnimation().

8.1.4.16 MaxStamina

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

Definition at line 26 of file BaseEnemy.cs.

Referenced by _Process(), _Ready(), and CheckPlayerHit().

8.1.4.17 Player

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

Definition at line 58 of file BaseEnemy.cs.

Referenced by OnHitboxAreaEntered(), and TakeDamage().

8.1.4.18 Pursuing

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

Definition at line 40 of file BaseEnemy.cs.

Referenced by HandleMovement(), OnDetectionBodyEntered(), OnPursuingRadiusBodyExited(), and TakeDamage().

8.1.4.19 Respawnable

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

Definition at line 20 of file BaseEnemy.cs.

8.1.4.20 ReturnToStart

```
double BaseEnemy.ReturnToStart  [protected]
```

Definition at line 39 of file BaseEnemy.cs.

Referenced by _Ready(), and HandleMovement().

8.1.4.21 ReturnToStartAfter

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

Definition at line 32 of file BaseEnemy.cs.

Referenced by _Ready(), and HandleMovement().

8.1.4.22 RightFallProtection

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

Definition at line 56 of file BaseEnemy.cs.

Referenced by _Ready(), and HandleMovement().

8.1.4.23 SinAmount

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

Definition at line 30 of file BaseEnemy.cs.

Referenced by Boss1._Ready(), and Die().

8.1.4.24 Speed

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

Definition at line 28 of file BaseEnemy.cs.

Referenced by Boss1._Ready(), and HandleMovement().

8.1.4.25 Sprite

```
AnimatedSprite2D BaseEnemy.Sprite  [protected]
```

Definition at line 49 of file BaseEnemy.cs.

Referenced by _Ready(), CheckPlayerHit(), Die(), FlipRotation(), HandleMovement(), OnSwordHitBoxBodyEntered(), Respawn(), Boss1.StartGlowing(), TakeDamage(), and UpdateAnimation().

8.1.4.26 StartPosition

```
Vector2 BaseEnemy.StartPosition  [protected]
```

Definition at line 43 of file BaseEnemy.cs.

Referenced by _Ready(), and HandleMovement().

8.1.4.27 StartRotation

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

Definition at line 44 of file BaseEnemy.cs.

Referenced by _Ready(), HandleMovement(), and SetRotation().

8.1.4.28 SwordHitbox

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

Definition at line 51 of file BaseEnemy.cs.

Referenced by _Ready(), CheckPlayerHit(), and FlipRotation().

8.1.4.29 TargetPosition

```
Vector2 BaseEnemy.TargetPosition = Vector2.Inf  [protected]
```

Definition at line 42 of file BaseEnemy.cs.

Referenced by HandleMovement().

8.1.5 Property Documentation

8.1.5.1 Id

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

Definition at line 34 of file BaseEnemy.cs.

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

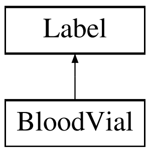
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BaseEnemy.cs

8.2 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.

8.2.1 Detailed Description

Klasse für die Interaktion zum heilen.

Definition at line 8 of file BloodVial.cs.

8.2.2 Member Function Documentation

8.2.2.1 _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     }
```

References PlayerStats.GetBVCurrentUses(), and PlayerStats.Instance.

8.2.2.2 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.

```
                                  {
        PlayerStats.Instance.SetBVMaxUses(PlayerStats.Instance.GetBVMaxUses() + Amount);
        ResetUses();
    }
```

References PlayerStats.GetBVMaxUses(), PlayerStats.Instance, ResetUses(), and PlayerStats.SetBVMaxUses().

8.2.2.3 LevelHealAmount()

`void BloodVial.LevelHealAmount ()  [inline]`

Verbessert den HealAMount eines Bloodvials um 25.

Definition at line 48 of file BloodVial.cs.

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

References PlayerStats.GetBVHealAmount(), PlayerStats.Instance, and PlayerStats.SetBVHealAmount().

8.2.2.4 ResetUses()

`void BloodVial.ResetUses ()  [inline]`

Setzt die Anzahl der Bloodvials auf das Maximum.

Definition at line 31 of file BloodVial.cs.

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

References PlayerStats.GetBVCurrentUses(), PlayerStats.GetBVMaxUses(), PlayerStats.Instance, and PlayerStats.SetBVCurrentUse

Referenced by AddMaxUses(), Checkpoint.OnPlayerBodyEntered(), and Player.Respawn().

8.2.2.5 UseBloodVial()

`void BloodVial.UseBloodVial ()  [inline]`

Versucht ein Bloodvial zu verwenden um den Spieler zu Heilen.

Definition at line 21 of file BloodVial.cs.

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

References PlayerStats.GetBVCurrentUses(), PlayerStats.GetBVHealAmount(), PlayerStats.GetCurrentHealth(), PlayerStats.Instance, PlayerStats.SetBVCurrentUses(), and PlayerStats.SetCurrentHealth().

Referenced by Player._PhysicsProcess().

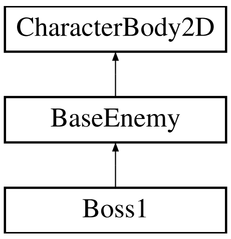
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BloodVial.cs

8.3 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.

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.
- bool IsDead ()

  Gibt boolean Dead zurück.
- void Respawn ()

  Wird aufgerufen wenn der Gegner respawnt.

Private Member Functions

- void HandleRegeneration (double DeltaTime)

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

  Startet einen Leuchteffekt, wenn der Boss Schaden nimmt.
- void ShowPopupMessage (string Message)

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

  Lässt alle toten Feinde im Raum des Bosses wiederbeleben.

Private Attributes

- bool EnemiesRevived = false
- 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 Dead = false
- bool Respawnable = true
- float MaxHealthPoints = 100f
- float Armor = 20f
- float MaxStamina = 1f
- float Speed = 10
- int SinAmount = 10
- double ReturnToStartAfter = 5
- float CurrentHealthPoints
- float CurrentStamina
- double ReturnToStart
- bool Pursuing = false
- Node2D CurrentTarget = null
- Vector2 TargetPosition = Vector2.Inf
- Vector2 StartPosition
- bool StartRotation = false
- bool AlreadyHit = false
- AnimatedSprite2D Sprite
- CollisionPolygon2D CollisionPolygon
- Area2D SwordHitbox
- CollisionShape2D MainCollision
- RayCast2D FrontCollisionRayCast
- RayCast2D LineOfSight
- RayCast2D LeftFallProtection
- RayCast2D RightFallProtection
- TextureProgressBar HealthBar
- Player Player

Properties inherited from BaseEnemy

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

8.3.1 Detailed Description

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

Definition at line 7 of file Boss1.cs.

8.3.2 Member Function Documentation

8.3.2.1 _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.

```
                                                      {
            base._Process(DeltaTime);

            if (CurrentHealthPoints <= MaxHealthPoints / 2 && !EnemiesRevived){
                StartGlowing();
                ReviveEnemies();
                EnemiesRevived = true;
                Armor = 60f; // Rüstung erhöhen
            }

            HandleRegeneration(DeltaTime);
        }
```

References BaseEnemy.Armor, BaseEnemy.CurrentHealthPoints, EnemiesRevived, HandleRegeneration(), BaseEnemy.MaxHealthPoints, ReviveEnemies(), and StartGlowing().

8.3.2.2 _Ready()

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

Überschreibt die _Ready-Methode von BaseEnemy.

Definition at line 18 of file Boss1.cs.

```
                                    {

            MaxHealthPoints = 400f;
            Damage = 50f;
            Armor = 30f;
            Speed = 10f;
            SinAmount = 100; // Bonuspunkte für Spieler beim Besiegen des Bosses

            base._Ready();

            CurrentHealthPoints = MaxHealthPoints;
            HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
        }
```

References BaseEnemy.Armor, BaseEnemy.CurrentHealthPoints, BaseEnemy.MaxHealthPoints, BaseEnemy.SinAmount, and BaseEnemy.Speed.

8.3.2.3 HandleRegeneration()

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

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.
```
                                                        {
        if (CurrentHealthPoints < MaxHealthPoints){
            RegenTimer += (float)DeltaTime;

            if (RegenTimer >= RegenCooldown){
                CurrentHealthPoints = Math.Min(CurrentHealthPoints + RegenAmount, MaxHealthPoints);
                HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
                RegenTimer = 0.0f; // Timer zurücksetzen
            }
        }
    }
```

References BaseEnemy.CurrentHealthPoints, BaseEnemy.MaxHealthPoints, RegenAmount, RegenCooldown, and RegenTimer.

Referenced by _Process().

8.3.2.4 ReviveEnemies()

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

Lässt alle toten Feinde im Raum des Bosses wiederbeleben.

Definition at line 108 of file Boss1.cs.
```
    {
        // Hole den Elternknoten (bossRoom)
        Node BossRoom = GetParent();

        // Iteriere durch alle Kinder von bossRoom
        foreach (Node Child in BossRoom.GetChildren()){
            if (Child is BaseEnemy BaseEnemy && BaseEnemy.IsDead()){
                BaseEnemy.Respawn();
            }
        }
    }
```

References BaseEnemy.IsDead(), and BaseEnemy.Respawn().

Referenced by _Process().

8.3.2.5 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.

```
                                              {
        Label popup = new Label();
        popup.Text = Message;
        popup.AddThemeColorOverride("font_color", new Color(1, 0, 0)); // Rot
        popup.Modulate = new Color(1, 1, 1, 0); // Start transparent
        popup.AutowrapMode = TextServer.AutowrapMode.Word;
        popup.SizeFlagsHorizontal = (Control.SizeFlags)(int)Control.SizeFlags.ExpandFill;
        popup.SizeFlagsVertical = (Control.SizeFlags)(int)Control.SizeFlags.ShrinkCenter;
        popup.HorizontalAlignment = HorizontalAlignment.Center;
        popup.VerticalAlignment = VerticalAlignment.Center;


        Vector2 bossGlobalPosition = GetGlobalTransformWithCanvas().Origin;
        popup.GlobalPosition = bossGlobalPosition + new Vector2(0, -100);

        CanvasLayer canvas = new CanvasLayer();
        AddChild(canvas);
        canvas.AddChild(popup);

        var tween = CreateTween();
        tween.TweenProperty(popup, "modulate:a", 1, 0.5f).From(0); // Einblenden
        tween.TweenProperty(popup, "modulate:a", 0, 0.5f).From(1).SetDelay(1.0f); // Ausblenden nach 1
   Sekunde
        tween.Connect("finished", new Callable(popup, "queue_free"));
    }
```

Referenced by StartGlowing().

8.3.2.6 StartGlowing()

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

Startet einen Leuchteffekt, wenn der Boss Schaden nimmt.

Definition at line 68 of file Boss1.cs.

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

References ShowPopupMessage(), and BaseEnemy.Sprite.

Referenced by _Process().

8.3.3 Member Data Documentation

8.3.3.1 EnemiesRevived

```
bool Boss1.EnemiesRevived = false  [private]
```

Definition at line 9 of file Boss1.cs.

Referenced by _Process().

8.3.3.2 RegenAmount

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

Definition at line 12 of file Boss1.cs.

Referenced by HandleRegeneration().

8.3.3.3 RegenCooldown

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

Definition at line 10 of file Boss1.cs.

Referenced by HandleRegeneration().

8.3.3.4 RegenTimer

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

Definition at line 11 of file Boss1.cs.

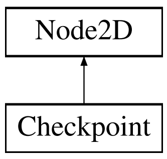
Referenced by HandleRegeneration().

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

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Boss1.cs

8.4 Checkpoint Class Reference

Inheritance diagram for Checkpoint:



Public Member Functions

- override void _Ready ()

Private Member Functions

- void OnPlayerBodyEntered (Node body)

Private Attributes

- Player Player

8.4.1 Detailed Description

Definition at line 4 of file Checkpoint.cs.

8.4.2 Member Function Documentation

8.4.2.1 _Ready()

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

Definition at line 13 of file Checkpoint.cs.

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

8.4.2.2 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.

```
    {

        if (body is Player Player)
        {
            // Setzen des Spawnpoints
            PlayerStats PlayerStats = GetNode<PlayerStats>("/root/PlayerStats");
            PlayerStats.Instance.SetSpawnPoint(this.GlobalPosition);
            Player.MaxHeal();
            PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetMaxStamina());
            Player.GetBloodVials().ResetUses();
            GD.Print("Spawnpoint des Players gesetzt auf: ", this.GlobalPosition);

            PlayerStats.SetRespawnLevelTag(GetParent().Name);
            GD.Print("RespawnLevelTag des Players gesetzt auf: ", GetParent().Name);
            GD.Print(PlayerStats.Instance.GetRespawnLevelTag());
        }

    }
```

References Player.GetBloodVials(), PlayerStats.GetMaxStamina(), PlayerStats.GetRespawnLevelTag(), PlayerStats.Instance, Player.MaxHeal(), BloodVial.ResetUses(), PlayerStats.SetRespawnLevelTag(), PlayerStats.SetSpawnPoint(), and PlayerStats.SetStamina().

8.4.3 Member Data Documentation

8.4.3.1 Player

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

Definition at line 8 of file Checkpoint.cs.

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

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Checkpoint.cs

8.5 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

8.5.1 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.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 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         }
```

References PhysicalDMG, PushAmount, Source, and TrueDMG.

8.5.3 Member Function Documentation

8.5.3.1 GetPhysicalDMG()

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

Gibt den physischen Schaden zurück.

Returns

Der physische Schaden.

Definition at line 31 of file Damage.cs.

```
                                  {
            return PhysicalDMG;
        }
```

References PhysicalDMG.

Referenced by BaseEnemy.TakeDamage(), and Player.TakeDamage().

8.5.3.2 GetPushAmount()

```
Vector2 Damage.GetPushAmount ()  [inline]
```

Gibt den Rückstoßvektor zurück.

Returns

Der Rückstoßvektor.

Definition at line 47 of file Damage.cs.

```
                                   {
            return PushAmount;
        }
```

References PushAmount.

Referenced by BaseEnemy.TakeDamage(), and Player.TakeDamage().

8.5.3.3 GetSource()

```
Node2D Damage.GetSource ()  [inline]
```

Gibt die Ursache zurück.

Returns

Die Ursache.

Definition at line 55 of file Damage.cs.

```
                              {
            return Source;
        }
```

References Source.

Referenced by BaseEnemy.TakeDamage().

8.5.3.4 GetTrueDMG()

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

Gibt den wahren Schaden zurück.

Returns

Der wahre Schaden.

Definition at line 39 of file Damage.cs.

```
                               {
        return TrueDMG;
    }
```

References TrueDMG.

Referenced by BaseEnemy.TakeDamage(), and Player.TakeDamage().

8.5.4 Member Data Documentation

8.5.4.1 PhysicalDMG

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

Definition at line 9 of file Damage.cs.

Referenced by Damage(), and GetPhysicalDMG().

8.5.4.2 PushAmount

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

Definition at line 11 of file Damage.cs.

Referenced by Damage(), and GetPushAmount().

8.5.4.3 Source

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

Definition at line 12 of file Damage.cs.

Referenced by Damage(), and GetSource().

8.5.4.4 TrueDMG

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

Definition at line 10 of file Damage.cs.

Referenced by Damage(), and GetTrueDMG().

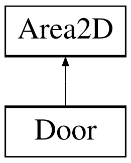
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Damage.cs

8.6 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.

8.6.1 Detailed Description

Klasse für die Tür.

Die Klasse ist für den Wechsel zwischen den Levels zuständig.

Definition at line 8 of file Door.cs.

8.6.2 Member Function Documentation

8.6.2.1 _Ready()

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

Initialisierung der Node Spawn.

Definition at line 26 of file Door.cs.

```
    {
        Spawn = GetNode("Spawn");
    }
```

References Spawn.

8.6.2.2 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.

```
    {
        if (body is Player player)
        {
            var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
            NavigationManager.GoToLevel(DestinationLevelTag, DestinationDoorTag);
        }
    }
```

References DestinationDoorTag, DestinationLevelTag, and NavigationManager.GoToLevel().

8.6.3 Member Data Documentation

8.6.3.1 Spawn

```
Node Door.Spawn
```

Definition at line 10 of file Door.cs.

Referenced by _Ready().

8.6.4 Property Documentation

8.6.4.1 DestinationDoorTag

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

Definition at line 16 of file Door.cs.

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

Referenced by OnPlayerBodyEntered().

8.6.4.2 DestinationLevelTag

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

Definition at line 13 of file Door.cs.

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

Referenced by OnPlayerBodyEntered().

8.6.4.3 SpawnDirection

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

Definition at line 19 of file Door.cs.

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

Referenced by LevelManager.OnLevelSpawn().

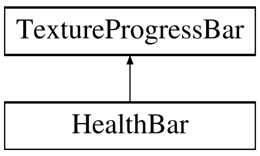
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Door.cs

8.7 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.

8.7.1 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.

8.7.2 Member Function Documentation

8.7.2.1 _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.

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

References PlayerStats.GetCurrentHealth(), and PlayerStats.Instance.

8.7.2.2 _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.

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

References PlayerStats.GetCurrentHealth(), PlayerStats.GetMaxHealthPoints(), and PlayerStats.Instance.

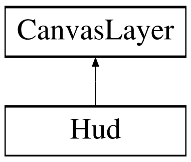
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/HealthBar.cs

8.8 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

8.8.1 Detailed Description

Klasse für das PauseMenu.

Definition at line 8 of file Hud.cs.

8.8.2 Member Function Documentation

8.8.2.1 _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     }
```

References TogglePause().

8.8.2.2 _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     }
```

References AnimationPlayer, and Buttons.

8.8.2.3 OnResumeButtonPressed()

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

Signal für den Resume-Button.

Definition at line 53 of file Hud.cs.

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

References TogglePause().

8.8.2.4 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     }
```

References StorageManager.GetLastSaveId(), StorageManager.Instance, and StorageManager.SaveAll().

8.8.2.5 OnSaveMenuButtonPressed()

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

Signal für den SaveAndReturnToMenu-Button.

Definition at line 67 of file Hud.cs.

```
                                        {
        StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
        NavigationManager.Instance.GoToLevel("main_menu", null);
        PlayerStats.Instance.Reload();
        GetTree().Paused = false;
    }
```

References StorageManager.GetLastSaveId(), NavigationManager.GoToLevel(), NavigationManager.Instance, PlayerStats.Instance, StorageManager.Instance, PlayerStats.Reload(), and StorageManager.SaveAll().

8.8.2.6 OnSaveQuitButtonPressed()

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

Signal für den SaveAndQuit-Button.

Definition at line 77 of file Hud.cs.

```
                                        {
        StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
        GetTree().Quit();
    }
```

References StorageManager.GetLastSaveId(), StorageManager.Instance, and StorageManager.SaveAll().

8.8.2.7 TogglePause()

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

Toggled die Pause Funktion.

Definition at line 38 of file Hud.cs.

```
                               {
        Enabled = !Enabled;
        GetTree().Paused = Enabled;
        if(Enabled){
            AnimationPlayer.Play("Pause");
            Buttons.Visible = true;
        } else {
            AnimationPlayer.PlayBackwards("Pause");
            Buttons.Visible = false;
        }
    }
```

References AnimationPlayer, and Enabled.

Referenced by _Process(), and OnResumeButtonPressed().

8.8.3 Member Data Documentation

8.8.3.1 AnimationPlayer

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

Definition at line 10 of file Hud.cs.

Referenced by _Ready(), and TogglePause().

8.8.3.2 Buttons

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

Definition at line 11 of file Hud.cs.

Referenced by _Ready().

8.8.3.3 Enabled

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

Definition at line 12 of file Hud.cs.

Referenced by TogglePause().

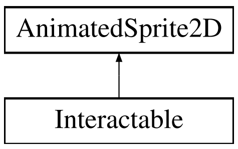
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Hud.cs

8.9 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

8.9.1 Detailed Description

Klasse für Interaktion.

Definition at line 7 of file Interactable.cs.

8.9.2 Member Function Documentation

8.9.2.1 _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.

```
                                                    {
        if(Input.IsActionJustPressed("interact")){
            Godot.Collections.Array<Node2D> Bodies = Area.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    TextLabel.Clear();
                    TextLabel.AppendText(Text);
                    PopUp.Visible = true;
                    return;
                }
            }
        }
    }
```

References Area, Text, and TextLabel.

8.9.2.2 _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.

```
                                {
        Player = GetNode<Player>("../Player");
        TextLabel = GetNode<RichTextLabel>("../HUD/PopUp/Text");
        PopUp = GetNode<Control>("../HUD/PopUp");
        Area = GetNode<Area2D>("Area2D");
    }
```

References Area, PopUp, and TextLabel.

8.9.2.3 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.

```
                                                        {
        if(Body == Player){
            PopUp.Visible = false;
            TextLabel.Clear();
        }
    }
```

References TextLabel.

8.9.3 Member Data Documentation

8.9.3.1 Area

`Area2D Interactable.Area  [private]`

Definition at line 12 of file Interactable.cs.

Referenced by _Process(), and _Ready().

8.9.3.2 Player

`Player Interactable.Player  [private]`

Definition at line 9 of file Interactable.cs.

8.9.3.3 PopUp

`Control Interactable.PopUp  [private]`

Definition at line 11 of file Interactable.cs.

Referenced by _Ready().

8.9.3.4 TextLabel

`RichTextLabel Interactable.TextLabel  [private]`

Definition at line 10 of file Interactable.cs.

Referenced by _Process(), _Ready(), and OnAreaBodyExited().

8.9.4 Property Documentation

8.9.4.1 Text

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

Definition at line 15 of file Interactable.cs.

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

Referenced by _Process().

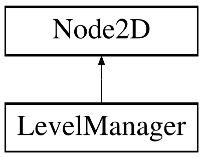
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Interactable.cs

8.10 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.

8.10.1 Detailed Description

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

Definition at line 7 of file LevelManager.cs.

8.10.2 Member Function Documentation

8.10.2.1 _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.

```
    {
        var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");

        if (NavigationManager.SpawnDoorTag != null)
        {
            OnLevelSpawn(NavigationManager.SpawnDoorTag);
        }
        else
        {
            NavigationManager.CallDeferred("TriggerPlayerSpawn", PlayerStats.Instance.GetPosition(), "");
        }

    }
```

References PlayerStats.GetPosition(), PlayerStats.Instance, OnLevelSpawn(), and NavigationManager.SpawnDoorTag.

8.10.2.2 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.

```
    {
        var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
        // Pfad zur Tür basierend auf dem Ziel-Tag erstellen
        string DoorPath = "Doors/Door_" + DestinationTag;

        Door door = GetNode<Door>(DoorPath);

        // TriggerPlayerSpawn nach deferred ausführen
        NavigationManager.CallDeferred("TriggerPlayerSpawn", door.GlobalPosition, door.SpawnDirection);
    }
```

References Door.SpawnDirection.

Referenced by _Ready().

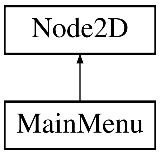
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/LevelManager.cs

8.11 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

8.11.1 Detailed Description

Klasse für das MainMenu.

Definition at line 7 of file MainMenu.cs.

8.11.2 Member Function Documentation

8.11.2.1 _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.

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

            SaveLabel[0] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Label");
            SelectButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Select");
            DeleteButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Delete");
            SaveLabel[1] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Label");
            SelectButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Select");
            DeleteButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Delete");
            SaveLabel[2] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Label");
            SelectButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Select");
            DeleteButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Delete");

            DeleteConfirmation = GetNode<ConfirmationDialog>("DeleteConfirmation");

            if(StorageManager.Instance.GetLastSaveId() > -1){
                ContinueButton.Visible = true;
            }
        }
```

References ContinueButton, DeleteButton, DeleteConfirmation, StorageManager.GetLastSaveId(), InfoLabel, StorageManager.Instance, Navigation, SaveLabel, SavesContainer, and SelectButton.

8.11.2.2 Change()

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

Wechselt das Menu zwischen den verschiedenen States.

Definition at line 52 of file MainMenu.cs.

```
                        {
        if(MenuState == 0){
            SavesContainer.Visible = false;
            Navigation.Visible = true;
        } else {
            Navigation.Visible = false;
            SavesContainer.Visible = true;

            int Saves = StorageManager.Instance.GetSaves();

            if(MenuState == 1){
                InfoLabel.Text = "Select empty save to start a new Game";
                for(int i = 0; i < 3; i++){
                    if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
                        SaveLabel[i].Text = "Save " + (i+1);
                        SelectButton[i].Disabled = true;
                        DeleteButton[i].Disabled = false;
                    } else {
                        SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
                        SelectButton[i].Disabled = false;
                        DeleteButton[i].Disabled = true;
                    }
                }
            } else {
                InfoLabel.Text = "Select save to load Game";
                for(int i = 0; i < 3; i++){
                    if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
                        SaveLabel[i].Text = "Save " + (i+1);
                        SelectButton[i].Disabled = false;
                        DeleteButton[i].Disabled = false;
                    } else {
                        SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
                        SelectButton[i].Disabled = true;
                        DeleteButton[i].Disabled = true;
                    }
                }
            }
        }
    }
```

References DeleteButton, StorageManager.GetSaves(), StorageManager.Instance, MenuState, SaveLabel, and SelectButton.

Referenced by OnBackButtonPressed(), OnDeleteConfirmationCanceled(), OnDeleteConfirmationConfirmed(), OnLoadGameButtonPressed(), and OnNewGameButtonPressed().

8.11.2.3 OnBackButtonPressed()

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

Signal für den Back-Button.

Definition at line 127 of file MainMenu.cs.

```
                                    {
        MenuState = 0;
        Change();
    }
```

References Change(), and MenuState.

8.11.2.4 OnContinueButtonPressed()

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

Signal für den Continue-Button.

Definition at line 95 of file MainMenu.cs.
```
                                       {
        StorageManager.Instance.LoadGameFile(StorageManager.Instance.GetLastSaveId());
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
    }
```

References PlayerStats.GetCurrentLevelTag(), StorageManager.GetLastSaveId(), NavigationManager.GoToLevel(), NavigationManager.Instance, PlayerStats.Instance, StorageManager.Instance, and StorageManager.LoadGameFile().

8.11.2.5 OnDeleteConfirmationCanceled()

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

Signal für den Delete-Abbruch.

Definition at line 198 of file MainMenu.cs.
```
                                            {
        SaveToDelete = 0;
        Change();
    }
```

References Change(), and SaveToDelete.

Referenced by OnDeleteConfirmationCloseRequested().

8.11.2.6 OnDeleteConfirmationCloseRequested()

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

Signal für das Schließen der Delete-Bestätigung.

Definition at line 214 of file MainMenu.cs.
```
                                                  {
        OnDeleteConfirmationCanceled();
    }
```

References OnDeleteConfirmationCanceled().

8.11.2.7 OnDeleteConfirmationConfirmed()

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

Signal für die Delete-Bestätigung.

Definition at line 206 of file MainMenu.cs.
```
                                             {
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() ^ (int) Math.Pow(2,
  SaveToDelete - 1));
        Change();
    }
```

References Change(), StorageManager.GetSaves(), StorageManager.Instance, SaveToDelete, and StorageManager.SetSaves().

8.11.2.8 OnLoadGameButtonPressed()

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

Signal für den LoadGame-Button.

Definition at line 119 of file MainMenu.cs.

```
                                        {
        MenuState = 2;
        Change();
    }
```

References Change(), and MenuState.

8.11.2.9 OnNewGameButtonPressed()

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

Signal für den NewGame-Button.

Definition at line 111 of file MainMenu.cs.

```
                                       {
        MenuState = 1;
        Change();
    }
```

References Change(), and MenuState.

8.11.2.10 OnQuitButtonPressed()

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

Signal für den Quit-Button.

Definition at line 103 of file MainMenu.cs.

```
                                    {
        StorageManager.Instance.SaveSettings();
        GetTree().Quit();
    }
```

References StorageManager.Instance, and StorageManager.SaveSettings().

8.11.2.11 OnSave1DeletePressed()

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

Signal für den Delete1-Button.

Definition at line 147 of file MainMenu.cs.

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

References DeleteConfirmation, and SaveToDelete.

8.11.2.12 OnSave1SelectPressed()

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

Signal für den Select1-Button.

Definition at line 135 of file MainMenu.cs.

```
                                 {
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(0);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 1);
        StorageManager.Instance.SetLastSaveId(0);
    }
```

References PlayerStats.GetCurrentLevelTag(), StorageManager.GetSaves(), NavigationManager.GoToLevel(), NavigationManager.Instance, PlayerStats.Instance, StorageManager.Instance, StorageManager.LoadGameFile(), MenuState, StorageManager.SetLastSaveId(), and StorageManager.SetSaves().

8.11.2.13 OnSave2DeletePressed()

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

Signal für den Delete2-Button.

Definition at line 168 of file MainMenu.cs.

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

References DeleteConfirmation, and SaveToDelete.

8.11.2.14 OnSave2SelectPressed()

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

Signal für den Select2-Button.

Definition at line 156 of file MainMenu.cs.

```
                                 {
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(1);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 2);
        StorageManager.Instance.SetLastSaveId(1);
    }
```

References PlayerStats.GetCurrentLevelTag(), StorageManager.GetSaves(), NavigationManager.GoToLevel(), NavigationManager.Instance, PlayerStats.Instance, StorageManager.Instance, StorageManager.LoadGameFile(), MenuState, StorageManager.SetLastSaveId(), and StorageManager.SetSaves().

8.11.2.15 OnSave3DeletePressed()

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

Signal für den Delete3-Button.

Definition at line 189 of file MainMenu.cs.

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

References DeleteConfirmation, and SaveToDelete.

8.11.2.16 OnSave3SelectPressed()

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

Signal für den Select3-Button.

Definition at line 177 of file MainMenu.cs.

```
                                    {
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(2);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 4);
        StorageManager.Instance.SetLastSaveId(2);
    }
```

References PlayerStats.GetCurrentLevelTag(), StorageManager.GetSaves(), NavigationManager.GoToLevel(), NavigationManager.Instance, PlayerStats.Instance, StorageManager.Instance, StorageManager.LoadGameFile(), MenuState, StorageManager.SetLastSaveId(), and StorageManager.SetSaves().

8.11.3 Member Data Documentation

8.11.3.1 ContinueButton

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

Definition at line 12 of file MainMenu.cs.

Referenced by _Ready().

8.11.3.2 DeleteButton

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

Definition at line 16 of file MainMenu.cs.

Referenced by _Ready(), and Change().

8.11.3.3 DeleteConfirmation

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

Definition at line 17 of file MainMenu.cs.

Referenced by _Ready(), OnSave1DeletePressed(), OnSave2DeletePressed(), and OnSave3DeletePressed().

8.11.3.4 InfoLabel

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

Definition at line 13 of file MainMenu.cs.

Referenced by _Ready().

8.11.3.5 MenuState

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

Definition at line 9 of file MainMenu.cs.

Referenced by Change(), OnBackButtonPressed(), OnLoadGameButtonPressed(), OnNewGameButtonPressed(), OnSave1SelectPressed(), OnSave2SelectPressed(), and OnSave3SelectPressed().

8.11.3.6 Navigation

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

Definition at line 10 of file MainMenu.cs.

Referenced by _Ready().

8.11.3.7 SaveLabel

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

Definition at line 14 of file MainMenu.cs.

Referenced by _Ready(), and Change().

8.11.3.8 SavesContainer

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

Definition at line 11 of file MainMenu.cs.

Referenced by _Ready().

8.11.3.9 SaveToDelete

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

Definition at line 18 of file MainMenu.cs.

Referenced by OnDeleteConfirmationCanceled(), OnDeleteConfirmationConfirmed(), OnSave1DeletePressed(), OnSave2DeletePressed(), and OnSave3DeletePressed().

8.11.3.10 SelectButton

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

Definition at line 15 of file MainMenu.cs.

Referenced by _Ready(), and Change().

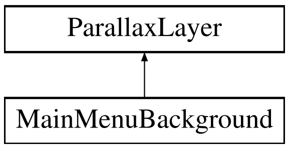
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/MainMenu.cs

8.12 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

8.12.1 Detailed Description

Klasse für die MainMenuBackground-Animation.

Definition at line 7 of file MainMenuBackground.cs.

8.12.2 Member Function Documentation

8.12.2.1 _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.

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

References ScrollSpeed.

8.12.3 Member Data Documentation

8.12.3.1 ScrollSpeed

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

Definition at line 10 of file MainMenuBackground.cs.

Referenced by _Process().

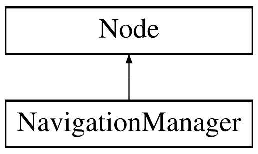
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/MainMenuBackground.cs

8.13 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")

8.13.1 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.

8.13.2 Member Function Documentation

8.13.2.1 _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     }
```

References Instance.

8.13.2.2 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.

```
    {
        GetTree().ChangeSceneToPacked(SceneToLoad);
    }
```

Referenced by GoToLevel().

8.13.2.3 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.

```
    {
        PackedScene SceneToLoad = null;

        // Bestimmen, welches Level geladen werden soll
        switch (LevelTag)
        {
            case "main_menu":
                SceneToLoad = SceneMainMenu;
                break;
            case "intro":
                SceneToLoad = SceneIntro;
                break;
            case "level1":
                SceneToLoad = SceneLevel1;
                break;
            case "bossRoom":
                SceneToLoad = SceneBoss;
                break;
            case "level_one":
                SceneToLoad = SceneLevelOne;
                break;
            case "level_two":
                SceneToLoad = SceneLevelTwo;
                break;
        }

        // Überprüfen, ob eine Szene ausgewählt wurde und diese dann laden
        if (SceneToLoad != null){
            if(SceneToLoad != SceneMainMenu){
                PlayerStats.Instance.SetCurrentLevelTag(LevelTag);
                SpawnDoorTag = DestinationTag;
            }
            // Verwendung der ChangeSceneToPacked-Methode in Godot 4
            CallDeferred(nameof(DeferredChangeScene), SceneToLoad);
        }
    }
```

References DeferredChangeScene(), PlayerStats.Instance, SceneBoss, SceneIntro, SceneLevel1, SceneLevelOne, SceneLevelTwo, SceneMainMenu, PlayerStats.SetCurrentLevelTag(), and SpawnDoorTag.

Referenced by MainMenu.OnContinueButtonPressed(), Door.OnPlayerBodyEntered(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), MainMenu.OnSave3SelectPressed(), Hud.OnSaveMenuButtonPressed(), and Player.Respawn().

8.13.2.4 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.

8.13.2.5 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.

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

8.13.3 Member Data Documentation

8.13.3.1 SceneBoss

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

Definition at line 14 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.3.2 SceneIntro

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

Definition at line 12 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.3.3 SceneLevel1

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

Definition at line 13 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.3.4 SceneLevelOne

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

Definition at line 15 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.3.5 SceneLevelTwo

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

Definition at line 16 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.3.6 SceneMainMenu

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

Definition at line 11 of file NavigationManager.cs.

Referenced by GoToLevel().

8.13.4 Property Documentation

8.13.4.1 Instance

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

Definition at line 9 of file NavigationManager.cs.

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

Referenced by _Ready(), MainMenu.OnContinueButtonPressed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), MainMenu.OnSave3SelectPressed(), and Hud.OnSaveMenuButtonPressed().

8.13.4.2 SpawnDoorTag

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

Definition at line 19 of file NavigationManager.cs.

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

Referenced by LevelManager._Ready(), and GoToLevel().

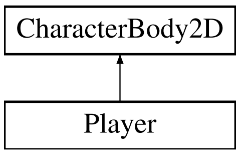
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/NavigationManager.cs

8.14 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.
- 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.
- bool IsAttacking ()

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

  Überprüft, ob der Spieler blockiert.
- 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

8.14.1 Detailed Description

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

Definition at line 8 of file Player.cs.

8.14.2 Member Function Documentation

8.14.2.1 _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.

```
                                                        {
        // Gravitation hinzufügen, wenn der Charakter nicht am Boden ist
        if (!IsOnFloor()) {
            Velocity += GetGravity() * (float)DeltaTime;
        } else {
            CanDash = true; // Dash wird zurückgesetzt, wenn der Charakter am Boden ist
        }

        TimeSinceLastStaminaUse += (float)DeltaTime;
        RegenerateStamina(20f, DeltaTime);

        // Heal
        if(Input.IsActionJustPressed("heal")){
            BloodVials.UseBloodVial();
        }

        HandleJump();
        HandleMovement(DeltaTime);
        MoveAndSlide();
        UpdateAnimations();
        PlayerStats.Instance.SetPosition(Position);
        }
```

References BloodVials, CanDash, HandleJump(), HandleMovement(), PlayerStats.Instance, RegenerateStamina(), PlayerStats.SetPosition(), TimeSinceLastStaminaUse, UpdateAnimations(), and BloodVial.UseBloodVial().

8.14.2.2 _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.

```
                                  {
        AnimationPlayer = GetNode<AnimationPlayer>("AnimationPlayer");
        Sprite = GetNode<Sprite2D>("Sprite2D");
        DashEffect = GetNode<Timer>("DashEffect");
        DashTimer = GetNode<Timer>("DashTimer");
        SwordCollision = GetNode<CollisionShape2D>("Sprite2D/SwordHit/SwordCollision");
        PlayerHitbox = GetNode<CollisionShape2D>("PlayerHitbox");
        HauptHitbox = PlayerHitbox.Position;
        BloodVials = GetNode<BloodVial>("../HUD/BloodVial/Counter");
        SinDisplay = GetNode<Label>("../HUD/SinAmount/Counter");

        SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";

        NavigationManager navigationManager = GetNode<NavigationManager>("/root/NavigationManager");
        navigationManager.Connect("OnTriggerPlayerSpawn", new Callable(this, nameof(OnSpawn)));

        Position = PlayerStats.Instance.GetPosition();
    }
```

References AnimationPlayer, BloodVials, DashEffect, DashTimer, PlayerStats.GetPosition(), PlayerStats.GetSinAmount(), HauptHitbox, PlayerStats.Instance, OnSpawn(), PlayerHitbox, SinDisplay, Sprite, and SwordCollision.

8.14.2.3 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     }
```

References DashTimer, and Sprite.

Referenced by DashInProgress().

8.14.2.4 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.

```
                                            {
        // Charakter bewegt sich in die Dash-Richtung mit Dash-Geschwindigkeit
        if (DashDirection == Vector2.Up) {
            Velocity = DashDirection / 1.5f * DashSpeed;
        } else {
            Velocity = DashDirection * DashSpeed;
        }

        // Dash-Trail bei Intervallen erstellen
        DashTrailTimer -= (float)DeltaTime;
        if (DashTrailTimer <= 0f) {
            CreateDashEffect();
            DashTrailTimer = DashTrailInterval;
        }
    }
```

References CreateDashEffect(), DashDirection, DashSpeed, DashTrailInterval, and DashTrailTimer.

Referenced by HandleMovement().

8.14.2.5 GetBloodVials()

`BloodVial Player.GetBloodVials ()  [inline]`

Getter für BloodVials.

Returns

BloodVial

Definition at line 383 of file Player.cs.

```
                              {
        return BloodVials;
    }
```

References BloodVials.

Referenced by Checkpoint.OnPlayerBodyEntered().

8.14.2.6 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.

```
                            {
        if(LastAttack == 1){
            return new Damage(50, 0, Vector2.Zero, this);
        }
        if(LastAttack == 2){
            Vector2 Push = new Vector2(20,0);
            if(Sprite.FlipH){
                Push = -Push;
            }
            return new Damage(100, 0, Push, this);
        }
        return new Damage(0,0,Vector2.Zero, this);
    }
```

References LastAttack, and Sprite.

Referenced by BaseEnemy.OnHitboxAreaEntered().

8.14.2.7 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.

```
                          {
        // Sprungzähler zurücksetzen, wenn der Charakter am Boden ist
        if (JumpCount != 0 && IsOnFloor()) {
            JumpCount = 0;
        }

        // Überprüfen, ob der Sprung-Button gedrückt wurde und der Charakter noch Sprünge übrig hat
        if (Input.IsActionJustPressed("ui_up") && JumpCount < JumpMax) {
            if (JumpCount == 0) {
            // Erster Sprung ohne Stamina-Verlust
            Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
            JumpCount += 1;
            } else if (JumpCount > 0) {
                // Beim Doppelsprung Stamina prüfen und abziehen
                if (UseStamina(15)) {
                    Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
                    JumpCount += 1;
                }
            }
        }
    }
```

References JUMP_VELOCITY, JumpCount, JumpMax, and UseStamina().

Referenced by _PhysicsProcess().

8.14.2.8 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.

```
                                               {
        Vector2 direction = new Vector2(Input.GetAxis("ui_left", "ui_right"), Input.GetAxis("ui_up",
    "ui_down")).Normalized();
        float currentSpeed = SPEED;

        // Sprite umdrehen basierend auf der Bewegungsrichtung und Kollision umdrehen
        if (direction.X < 0) {
            Sprite.FlipH = true;
            SwordCollision.Position = new Vector2(-Mathf.Abs(SwordCollision.Position.X),
    SwordCollision.Position.Y);
            PlayerHitbox.Position = new Vector2(Sprite.Position.X * 1.8f, PlayerHitbox.Position.Y);
        } else if (direction.X > 0) {
            Sprite.FlipH = false;
            SwordCollision.Position = new Vector2(Mathf.Abs(SwordCollision.Position.X),
    SwordCollision.Position.Y);
            PlayerHitbox.Position = HauptHitbox;
        }

        // Geschwindigkeit reduzieren, wenn der Spieler angreift
        if (AnimationPlayer.CurrentAnimation == "light_attack") {
            currentSpeed *= 0.5f;
        } else if (AnimationPlayer.CurrentAnimation == "heavy_attack") {
            currentSpeed *= 0.15f;
        }

        // Blockieren stoppt die Bewegung
        if (IsBlocking()) {
            currentSpeed = 0;
        }

        if (IsDashing) {
            DashInProgress(DeltaTime);
        } else {
            // Normale Bewegung verarbeiten, wenn kein Dash aktiv ist
            if (direction != Vector2.Zero) {
                Velocity = new Vector2(direction.X * currentSpeed, Velocity.Y);
            } else {
                Velocity = new Vector2(Mathf.MoveToward(Velocity.X, 0, SPEED), Velocity.Y);
            }

            // Überprüfen, ob der Dash-Button gedrückt wurde mit eine Bewegungsrichtung und nicht
schon am angreifen ist
            if (Input.IsActionJustPressed("dash") && direction != Vector2.Zero && CanDash &&
!IsAttacking()) {
                // Wenn der Player genug Stamina hat kann er dashen
                if (UseStamina(20)){
                    DashDirection = direction;
                    StartDash();
                }
            }
        }
    }
```

References AnimationPlayer, CanDash, DashDirection, DashInProgress(), HauptHitbox, IsAttacking(), IsBlocking(), IsDashing, PlayerHitbox, SPEED, Sprite, StartDash(), SwordCollision, and UseStamina().

Referenced by _PhysicsProcess().

8.14.2.9 IsAttacking()

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

Überprüft, ob der Spieler gerade angreift.

Returns

true, wenn der Spieler angreift.

Definition at line 265 of file Player.cs.

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

Referenced by HandleMovement(), and UpdateAnimations().

8.14.2.10 IsBlocking()

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

Überprüft, ob der Spieler blockiert.

Returns

true, wenn der Spieler blockiert.

Definition at line 273 of file Player.cs.

```
                             {
        return AnimationPlayer.CurrentAnimation == "block";
    }
```

Referenced by HandleMovement(), TakeDamage(), and UpdateAnimations().

8.14.2.11 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.

```
                    {
        PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetMaxHealthPoints());
    }
```

References PlayerStats.GetMaxHealthPoints(), PlayerStats.Instance, and PlayerStats.SetCurrentHealth().

Referenced by Checkpoint.OnPlayerBodyEntered().

8.14.2.12 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.

```
                                                         {

        // Spielerposition auf die übergebene Position setzen
        if (direction == "right")
        {
            // Update the x value by adding 50 to it, keep the original y value
            Sprite.FlipH = false;
            position = position with { X = position.X + 25 };
        }
        else if (direction == "left")
        {
            // Update the x value by subtracting 50 from it, keep the original y value
            Sprite.FlipH = true;
            position = position with { X = position.X - 25 };
        }
        Position = position;

    }
```

Referenced by _Ready().

8.14.2.13 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     }
```

References PlayerStats.GetStamina(), PlayerStats.Instance, PlayerStats.SetStamina(), and TimeSinceLastStaminaUse.

Referenced by _PhysicsProcess().

8.14.2.14 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     }
```

References BloodVials, PlayerStats.GetRespawnLevelTag(), NavigationManager.GoToLevel(), PlayerStats.Instance, and BloodVial.ResetUses().

Referenced by TakeDamage().

8.14.2.15 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     }
```

References PlayerStats.GetSinAmount(), PlayerStats.Instance, and PlayerStats.SetSinAmount().

Referenced by BaseEnemy.Die().

8.14.2.16 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     }
```

Referenced by Spike.OnPlayerBodyEntered(), and SpikeDynamic.OnPlayerBodyEntered().

8.14.2.17 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     }
```

References CanDash, DashEffect, DashTimer, DashTrailTimer, IsDashing, and StopDash().

Referenced by HandleMovement().

8.14.2.18 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     }
```

References DashEffect, DashTimer, IsDashing, and StopDash().

Referenced by StartDash(), and StopDash().

8.14.2.19 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.

```
                                        {
        float totalDamage = Damage.GetTrueDMG();
        if(!IsBlocking()){
            totalDamage += Damage.GetPhysicalDMG();
        } else {
            float CurrentStamina = PlayerStats.Instance.GetStamina();
            CurrentStamina -= Damage.GetPhysicalDMG();
            if(CurrentStamina < 0){
                totalDamage -= CurrentStamina;
            }
            PlayerStats.Instance.SetStamina(CurrentStamina);
        }

        PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() - totalDamage);
        Position += Damage.GetPushAmount();

        // Überprüfe, ob der Spieler gestorben ist
        if (PlayerStats.Instance.GetCurrentHealth() <= 0){
            GD.Print("Spieler ist gestorben!");
            Respawn();
        }
    }
```

References PlayerStats.GetCurrentHealth(), Damage.GetPhysicalDMG(), Damage.GetPushAmount(), PlayerStats.GetStamina(), Damage.GetTrueDMG(), PlayerStats.Instance, IsBlocking(), Respawn(), PlayerStats.SetCurrentHealth(), and PlayerStats.SetStamina().

Referenced by BaseEnemy.CheckPlayerHit(), Spike.OnPlayerBodyEntered(), SpikeDynamic.OnPlayerBodyEntered(), Spike.OnTimerTimeout(), and SpikeDynamic.OnTimerTimeout().

8.14.2.20 UpdateAnimations()

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

Aktualisiert die Animationen des Spielers.

Definition at line 425 of file Player.cs.

```
                                    {
        if (Input.IsActionJustPressed("light_attack") && !IsDashing && !IsAttacking()) {
            if (UseStamina(10)){
                LastAttack = 1;
                AnimationPlayer.Play("light_attack");
            }
        } else if (Input.IsActionJustPressed("heavy_attack") && !IsDashing && !IsAttacking()) {
            if (UseStamina(25)){
                LastAttack = 2;
                AnimationPlayer.Play("heavy_attack");
            }
        }
        if (Input.IsActionPressed("block") && !IsDashing && !IsAttacking() && IsOnFloor()) {
            if (UseStamina(0)){
                AnimationPlayer.Play("block");
                LastAttack = 0;
            }
        }

        if (IsOnFloor() && !IsAttacking() && !IsBlocking()) {
            LastAttack = 0;
            if (Velocity.X == 0) {
                AnimationPlayer.Play("idle");
            } else {
                AnimationPlayer.Play("run");
            }
        } else if (!IsOnFloor() && !IsAttacking() && !IsBlocking()) {
            LastAttack = 0;
            if (Velocity.Y < 0) {
                AnimationPlayer.Play("jump");
            } else if (Velocity.Y > 0) {
                AnimationPlayer.Play("fall");
            }
        }
    }
```

References AnimationPlayer, IsAttacking(), IsBlocking(), IsDashing, LastAttack, and UseStamina().

Referenced by _PhysicsProcess().

8.14.2.21 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.

```
                                        {
        // Versucht, eine bestimmte Menge an Stamina zu verbrauchen.
        // Gibt true zurück, wenn genug Stamina verfügbar war; andernfalls false.
        if (PlayerStats.Instance.GetStamina() >= Amount) {
            PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() - Amount);
            TimeSinceLastStaminaUse = 0f;
            return true;
        }

        return false;
    }
```

References PlayerStats.GetStamina(), PlayerStats.Instance, PlayerStats.SetStamina(), and TimeSinceLastStaminaUse.

Referenced by HandleJump(), HandleMovement(), and UpdateAnimations().

8.14.3 Member Data Documentation

8.14.3.1 AnimationPlayer

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

Definition at line 24 of file Player.cs.

Referenced by _Ready(), HandleMovement(), and UpdateAnimations().

8.14.3.2 BloodVials

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

Definition at line 30 of file Player.cs.

Referenced by _PhysicsProcess(), _Ready(), GetBloodVials(), and Respawn().

8.14.3.3 CanDash

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

Definition at line 19 of file Player.cs.

Referenced by _PhysicsProcess(), HandleMovement(), and StartDash().

8.14.3.4 DashDirection

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

Definition at line 16 of file Player.cs.

Referenced by DashInProgress(), and HandleMovement().

8.14.3.5 DashEffect

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

Definition at line 26 of file Player.cs.

Referenced by _Ready(), StartDash(), and StopDash().

8.14.3.6 DashSpeed

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

Definition at line 17 of file Player.cs.

Referenced by DashInProgress().

8.14.3.7 DashTimer

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

Definition at line 27 of file Player.cs.

Referenced by _Ready(), CreateDashEffect(), StartDash(), and StopDash().

8.14.3.8 DashTrailInterval

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

Definition at line 20 of file Player.cs.

Referenced by DashInProgress().

8.14.3.9 DashTrailTimer

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

Definition at line 21 of file Player.cs.

Referenced by DashInProgress(), and StartDash().

8.14.3.10 HauptHitbox

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

Definition at line 33 of file Player.cs.

Referenced by _Ready(), and HandleMovement().

8.14.3.11 IsDashing

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

Definition at line 18 of file Player.cs.

Referenced by HandleMovement(), StartDash(), StopDash(), and UpdateAnimations().

8.14.3.12 JUMP_VELOCITY

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

Definition at line 12 of file Player.cs.

Referenced by HandleJump().

8.14.3.13 JumpCount

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

Definition at line 14 of file Player.cs.

Referenced by HandleJump().

8.14.3.14 JumpMax

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

Definition at line 13 of file Player.cs.

Referenced by HandleJump().

8.14.3.15 LastAttack

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

Definition at line 34 of file Player.cs.

Referenced by GetDamage(), and UpdateAnimations().

8.14.3.16 PlayerHitbox

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

Definition at line 29 of file Player.cs.

Referenced by _Ready(), and HandleMovement().

8.14.3.17 SinDisplay

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

Definition at line 31 of file Player.cs.

Referenced by _Ready().

8.14.3.18 SPEED

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

Definition at line 11 of file Player.cs.

Referenced by HandleMovement().

8.14.3.19 Sprite

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

Definition at line 25 of file Player.cs.

Referenced by _Ready(), CreateDashEffect(), GetDamage(), and HandleMovement().

8.14.3.20 SwordCollision

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

Definition at line 28 of file Player.cs.

Referenced by _Ready(), and HandleMovement().

8.14.3.21 TimeSinceLastStaminaUse

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

Definition at line 37 of file Player.cs.

Referenced by _PhysicsProcess(), RegenerateStamina(), and UseStamina().

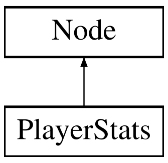
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Player.cs

8.15 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

8.15.1 Detailed Description

Klasse für die Spielerstats.

Definition at line 7 of file PlayerStats.cs.

8.15.2 Member Function Documentation

8.15.2.1 _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     }
```

References BVCurrentUses, BVMaxUses, CurrentHealth, CurrentStamina, Instance, MaxHealthPoints, and MaxStamina.

Referenced by Reload().

8.15.2.2 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     }
```

References BVCurrentUses.

Referenced by BloodVial._Ready(), BloodVial.ResetUses(), StorageManager.SaveGameFile(), and BloodVial.UseBloodVial().

8.15.2.3 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     }
```

References BVHealAmount.

Referenced by BloodVial.LevelHealAmount(), StorageManager.SaveGameFile(), and BloodVial.UseBloodVial().

8.15.2.4 GetBVMaxUses()

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

Gibt die MaxUses der Bloodvials zurück.

Returns

Die aktuellen MaxUses.

Definition at line 214 of file PlayerStats.cs.

```
                             {
        return BVMaxUses;
    }
```

References BVMaxUses.

Referenced by BloodVial.AddMaxUses(), BloodVial.ResetUses(), and StorageManager.SaveGameFile().

8.15.2.5 GetCurrentHealth()

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

Gibt die aktuellen Lebenspunkte des Spielers zurück.

Returns

Die aktuellen Lebenspunkte.

Definition at line 139 of file PlayerStats.cs.

```
                                    {
        return CurrentHealth;
    }
```

References CurrentHealth.

Referenced by HealthBar._Process(), HealthBar._Ready(), StorageManager.SaveGameFile(), Player.TakeDamage(), and BloodVial.UseBloodVial().

8.15.2.6 GetCurrentLevelTag()

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

Getter für CurrentLevelTag.

Returns

String CurrentLevelTag

Definition at line 56 of file PlayerStats.cs.

```
                                         {
        return CurrentLevelTag;
    }
```

References CurrentLevelTag.

Referenced by MainMenu.OnContinueButtonPressed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), MainMenu.OnSave3SelectPressed(), and StorageManager.SaveGameFile().

8.15.2.7 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     }
```

References MaxHealthPoints.

Referenced by HealthBar._Ready(), Player.MaxHeal(), and StorageManager.SaveGameFile().

8.15.2.8 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     }
```

References MaxStamina.

Referenced by StaminaBar._Ready(), Checkpoint.OnPlayerBodyEntered(), and StorageManager.SaveGameFile().

8.15.2.9 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     }
```

References Position.

Referenced by LevelManager._Ready(), Player._Ready(), and StorageManager.SaveGameFile().

8.15.2.10 GetRespawnLevelTag()

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

Getter für RespawnLevelTag.

Returns

String RespawnLevelTag

Definition at line 40 of file PlayerStats.cs.

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

References RespawnLevelTag.

Referenced by Checkpoint.OnPlayerBodyEntered(), Player.Respawn(), and StorageManager.SaveGameFile().

8.15.2.11 GetSinAmount()

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

Getter für SinAmount.

Returns

int Sins

Definition at line 105 of file PlayerStats.cs.

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

References SinAmount.

Referenced by Player._Ready(), BaseEnemy.Die(), StorageManager.SaveGameFile(), and Player.SetSinAmount().

8.15.2.12 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     }
```

References SpawnPoint.

Referenced by StorageManager.SaveGameFile().

8.15.2.13 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     }
```

References CurrentStamina.

Referenced by StaminaBar._Process(), StaminaBar._Ready(), Player.RegenerateStamina(), StorageManager.SaveGameFile(), Player.TakeDamage(), and Player.UseStamina().

8.15.2.14 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     }
```

References _Ready(), and Instance.

Referenced by Hud.OnSaveMenuButtonPressed().

8.15.2.15 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     }
```

References BVCurrentUses.

Referenced by StorageManager.LoadGameFile(), BloodVial.ResetUses(), and BloodVial.UseBloodVial().

8.15.2.16 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.

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

References BVHealAmount.

Referenced by BloodVial.LevelHealAmount(), and StorageManager.LoadGameFile().

8.15.2.17 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.

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

References BVMaxUses.

Referenced by BloodVial.AddMaxUses(), and StorageManager.LoadGameFile().

8.15.2.18 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.

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

References CurrentHealth, and MaxHealthPoints.

Referenced by StorageManager.LoadGameFile(), Player.MaxHeal(), Player.TakeDamage(), and BloodVial.UseBloodVial().

8.15.2.19 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     }
```

References CurrentLevelTag.

Referenced by NavigationManager.GoToLevel(), and StorageManager.LoadGameFile().

8.15.2.20 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     }
```

References MaxHealthPoints.

Referenced by StorageManager.LoadGameFile().

8.15.2.21 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     }
```

References MaxStamina.

Referenced by StorageManager.LoadGameFile().

8.15.2.22 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     }
```

References Position.

Referenced by Player._PhysicsProcess(), and StorageManager.LoadGameFile().

8.15.2.23 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     }
```

References RespawnLevelTag.

Referenced by StorageManager.LoadGameFile(), and Checkpoint.OnPlayerBodyEntered().

8.15.2.24 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     }
```

References SinAmount.

Referenced by StorageManager.LoadGameFile(), and Player.SetSinAmount().

8.15.2.25 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.

```
                                                 {
        SpawnPoint = spawnPoint;
    }
```

References SpawnPoint.

Referenced by StorageManager.LoadGameFile(), and Checkpoint.OnPlayerBodyEntered().

8.15.2.26 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.

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

References CurrentStamina, and MaxStamina.

Referenced by StorageManager.LoadGameFile(), Checkpoint.OnPlayerBodyEntered(), Player.RegenerateStamina(), Player.TakeDamage(), and Player.UseStamina().

8.15.3 Member Data Documentation

8.15.3.1 BVCurrentUses

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

Definition at line 23 of file PlayerStats.cs.

Referenced by _Ready(), GetBVCurrentUses(), and SetBVCurrentUses().

8.15.3.2 BVHealAmount

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

Definition at line 21 of file PlayerStats.cs.

Referenced by GetBVHealAmount(), and SetBVHealAmount().

8.15.3.3 BVMaxUses

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

Definition at line 22 of file PlayerStats.cs.

Referenced by _Ready(), GetBVMaxUses(), and SetBVMaxUses().

8.15.3.4 CurrentHealth

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

Definition at line 18 of file PlayerStats.cs.

Referenced by _Ready(), GetCurrentHealth(), and SetCurrentHealth().

8.15.3.5 CurrentLevelTag

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

Definition at line 13 of file PlayerStats.cs.

Referenced by GetCurrentLevelTag(), and SetCurrentLevelTag().

8.15.3.6 CurrentStamina

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

Definition at line 20 of file PlayerStats.cs.

Referenced by _Ready(), GetStamina(), and SetStamina().

8.15.3.7 MaxHealthPoints

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

Definition at line 17 of file PlayerStats.cs.

Referenced by _Ready(), GetMaxHealthPoints(), SetCurrentHealth(), and SetMaxHealthPoints().

8.15.3.8 MaxStamina

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

Definition at line 19 of file PlayerStats.cs.

Referenced by _Ready(), GetMaxStamina(), SetMaxStamina(), and SetStamina().

8.15.3.9 Position

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

Definition at line 15 of file PlayerStats.cs.

Referenced by GetPosition(), and SetPosition().

8.15.3.10 RespawnLevelTag

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

Definition at line 12 of file PlayerStats.cs.

Referenced by GetRespawnLevelTag(), and SetRespawnLevelTag().

8.15.3.11 SinAmount

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

Definition at line 16 of file PlayerStats.cs.

Referenced by GetSinAmount(), and SetSinAmount().

8.15.3.12 SpawnPoint

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

Definition at line 14 of file PlayerStats.cs.

Referenced by GetSpawnPoint(), and SetSpawnPoint().

8.15.4 Property Documentation

8.15.4.1 Instance

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

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

Referenced by Player._PhysicsProcess(), HealthBar._Process(), StaminaBar._Process(), BloodVial._Ready(), HealthBar._Ready(), LevelManager._Ready(), Player._Ready(), _Ready(), StaminaBar._Ready(), BloodVial.AddMaxUses(), BaseEnemy.Die(), NavigationManager.GoToLevel(), BloodVial.LevelHealAmount(), StorageManager.LoadGameFile(), Player.MaxHeal(), MainMenu.OnContinueButtonPressed(), Checkpoint.OnPlayerBodyEntered(), MainMenu.OnSave1SelectPressed() MainMenu.OnSave2SelectPressed(), MainMenu.OnSave3SelectPressed(), Hud.OnSaveMenuButtonPressed(), Player.RegenerateStamina(), Reload(), BloodVial.ResetUses(), Player.Respawn(), StorageManager.SaveGameFile(), Player.SetSinAmount(), Player.TakeDamage(), BloodVial.UseBloodVial(), and Player.UseStamina().

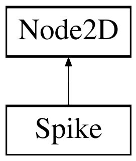
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/PlayerStats.cs

8.16 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

8.16.1 Detailed Description

Klasse für die Spikes.

Definition at line 7 of file Spike.cs.

8.16.2 Member Function Documentation

8.16.2.1 _Ready()

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

Initialisierung der Node Player.

Hier wird der Player Node gefunden

Definition at line 20 of file Spike.cs.

```
    {
        // Zugriff auf Player Node

        Player = GetNode<Player>("../../Player");
    }
```

8.16.2.2 GetDamage()

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

Gibt ein Damage Objekt zurück.

Returns

Damage Objekt

Definition at line 71 of file Spike.cs.

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

References Damage.

Referenced by OnPlayerBodyEntered(), and OnTimerTimeout().

8.16.2.3 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.

```
    {

        if (body is Player)
        {
            Player = (Player)body; // Instanzvariable setzen
            Player.TakeDamage(GetDamage());
            Player.SlowPlayer(0.5f);
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
            GD.Print("Player entered spike");
        }


    }
```

References GetDamage(), Player, Player.SlowPlayer(), and Player.TakeDamage().

8.16.2.4 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.

```
    {
        if (body is Player)
        {
            Player = null; // Instanzvariable zurücksetzen
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
        }
    }
```

8.16.2.5 OnTimerTimeout()

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

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

Definition at line 60 of file Spike.cs.

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

References GetDamage(), and Player.TakeDamage().

8.16.3 Member Data Documentation

8.16.3.1 Damage

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

Definition at line 14 of file Spike.cs.

Referenced by GetDamage().

8.16.3.2 Player

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

Definition at line 10 of file Spike.cs.

Referenced by OnPlayerBodyEntered().

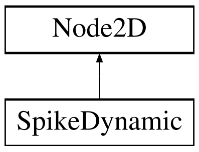
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Spike.cs

8.17 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

8.17.1 Detailed Description

Klasse für die beweglichen Spikes.

Definition at line 7 of file SpikeDynamic.cs.

8.17.2 Member Function Documentation

8.17.2.1 _Ready()

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

Initialisierung der Node Player.

Hier wird der Player Node gefunden

Definition at line 20 of file SpikeDynamic.cs.

```
    {
        // Zugriff auf Player Node

        Player = GetNode<Player>("../../../Player");
    }
```

8.17.2.2 GetDamage()

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

Gibt ein Damage Objekt zurück.

Returns

Damage Objekt

Definition at line 71 of file SpikeDynamic.cs.

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

References Damage.

Referenced by OnPlayerBodyEntered(), and OnTimerTimeout().

8.17.2.3 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.

```
    {

        if (body is Player)
        {
            Player = (Player)body; // Instanzvariable setzen
            Player.TakeDamage(GetDamage());
            Player.SlowPlayer(0.5f);
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
            GD.Print("Player entered spike");
        }


    }
```

References GetDamage(), Player, Player.SlowPlayer(), and Player.TakeDamage().

8.17.2.4 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.

```
    {
        if (body is Player)
        {
            Player = null; // Instanzvariable zurücksetzen
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
        }
    }
```

8.17.2.5 OnTimerTimeout()

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

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

Definition at line 60 of file SpikeDynamic.cs.

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

References GetDamage(), and Player.TakeDamage().

8.17.3 Member Data Documentation

8.17.3.1 Damage

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

Definition at line 13 of file SpikeDynamic.cs.

Referenced by GetDamage().

8.17.3.2 Player

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

Definition at line 10 of file SpikeDynamic.cs.

Referenced by OnPlayerBodyEntered().

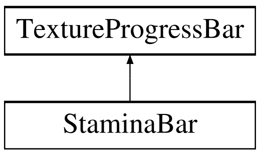
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/SpikeDynamic.cs

8.18 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.

8.18.1 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.

8.18.2 Member Function Documentation

8.18.2.1 _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     }
```

References PlayerStats.GetStamina(), and PlayerStats.Instance.

8.18.2.2 _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     }
```

References PlayerStats.GetMaxStamina(), PlayerStats.GetStamina(), and PlayerStats.Instance.

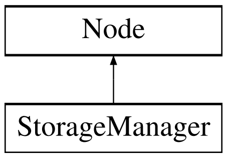
The documentation for this class was generated from the following file:

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/StaminaBar.cs

8.19 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"

8.19.1 Detailed Description

Klasse für das Speichern und Laden von Daten.

Definition at line 8 of file StorageManager.cs.

8.19.2 Member Function Documentation

8.19.2.1 _Ready()

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

Initialisierung der Instanz und erstes laden der Einstellungen.

Definition at line 20 of file StorageManager.cs.

```
                                    {
        LoadSettings();
        Instance = this;
    }
```

References Instance, and LoadSettings().

8.19.2.2 GetLastSaveId()

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

Getter für LastSaveId.

Returns

int LastSaveId

Definition at line 118 of file StorageManager.cs.

```
                                {
        return LastSaveId;
    }
```

References LastSaveId.

Referenced by MainMenu._Ready(), MainMenu.OnContinueButtonPressed(), Hud.OnSaveButtonPressed(), Hud.OnSaveMenuButtonPressed(), and Hud.OnSaveQuitButtonPressed().

8.19.2.3 GetSaves()

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

Getter für Saves.

Returns

int Saves

Definition at line 134 of file StorageManager.cs.

```
                           {
        return Saves;
    }
```

References Saves.

Referenced by MainMenu.Change(), MainMenu.OnDeleteConfirmationConfirmed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), and MainMenu.OnSave3SelectPressed().

8.19.2.4 LoadGameFile()

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

Laden eines Spielstandes.

Parameters

Id	des Spielstandes.

Definition at line 43 of file StorageManager.cs.

```
                                 {
        if(!FileAccess.FileExists(PathSave[id])){
            return;
        }
        FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Read);
        PlayerStats.Instance.SetRespawnLevelTag((String) File.GetVar());
        PlayerStats.Instance.SetCurrentLevelTag((String) File.GetVar());
        PlayerStats.Instance.SetSpawnPoint((Vector2) File.GetVar());
        PlayerStats.Instance.SetPosition((Vector2) File.GetVar());
        PlayerStats.Instance.SetSinAmount((int) File.GetVar());
        PlayerStats.Instance.SetMaxHealthPoints((float) File.GetVar());
        PlayerStats.Instance.SetCurrentHealth((float) File.GetVar());
        PlayerStats.Instance.SetMaxStamina((float) File.GetVar());
        PlayerStats.Instance.SetStamina((float) File.GetVar());
        PlayerStats.Instance.SetBVHealAmount((int) File.GetVar());
        PlayerStats.Instance.SetBVMaxUses((int) File.GetVar());
        PlayerStats.Instance.SetBVCurrentUses((int) File.GetVar());

        File.Close();
    }
```

References PlayerStats.Instance, PathSave, PlayerStats.SetBVCurrentUses(), PlayerStats.SetBVHealAmount(), PlayerStats.SetBVMaxUses(), PlayerStats.SetCurrentHealth(), PlayerStats.SetCurrentLevelTag(), PlayerStats.SetMaxHealthPoints(), PlayerStats.SetMaxStamina(), PlayerStats.SetPosition(), PlayerStats.SetRespawnLevelTag(), PlayerStats.SetSinAmount(), PlayerStats.SetSpawnPoint(), and PlayerStats.SetStamina().

Referenced by MainMenu.OnContinueButtonPressed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), and MainMenu.OnSave3SelectPressed().

8.19.2.5 LoadSettings()

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

Laden der Einstellungen.

Definition at line 28 of file StorageManager.cs.

```
                           {
        if(!FileAccess.FileExists(PathSettings)){
            return;
        }
        FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Read);
        Saves = (int) File.GetVar();
        LastSaveId = (int) File.GetVar();

        File.Close();
    }
```

References LastSaveId, PathSettings, and Saves.

Referenced by _Ready().

8.19.2.6 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     }
```

References SaveGameFile(), and SaveSettings().

Referenced by Hud.OnSaveButtonPressed(), Hud.OnSaveMenuButtonPressed(), and Hud.OnSaveQuitButtonPressed().

8.19.2.7 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     }
```

References PlayerStats.GetBVCurrentUses(), PlayerStats.GetBVHealAmount(), PlayerStats.GetBVMaxUses(), PlayerStats.GetCurrentHealth(), PlayerStats.GetCurrentLevelTag(), PlayerStats.GetMaxHealthPoints(), PlayerStats.GetMaxStamina() PlayerStats.GetPosition(), PlayerStats.GetRespawnLevelTag(), PlayerStats.GetSinAmount(), PlayerStats.GetSpawnPoint(), PlayerStats.GetStamina(), PlayerStats.Instance, and PathSave.

Referenced by SaveAll().

8.19.2.8 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     }
```

References LastSaveId, PathSettings, and Saves.

Referenced by MainMenu.OnQuitButtonPressed(), and SaveAll().

8.19.2.9 SetLastSaveId()

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

Setter für LastSaveId.

Parameters

int	Last↩ SaveId

Definition at line 110 of file StorageManager.cs.

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

References LastSaveId.

Referenced by MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), and MainMenu.OnSave3SelectPressed().

8.19.2.10 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     }
```

References Saves.

Referenced by MainMenu.OnDeleteConfirmationConfirmed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPresse
and MainMenu.OnSave3SelectPressed().

8.19.3 Member Data Documentation

8.19.3.1 LastSaveId

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

Definition at line 13 of file StorageManager.cs.

Referenced by GetLastSaveId(), LoadSettings(), SaveSettings(), and SetLastSaveId().

8.19.3.2 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"};
```

Referenced by LoadGameFile(), and SaveGameFile().

8.19.3.3 PathSettings

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

Definition at line 11 of file StorageManager.cs.

Referenced by LoadSettings(), and SaveSettings().

8.19.3.4 Saves

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

Definition at line 14 of file StorageManager.cs.

Referenced by GetSaves(), LoadSettings(), SaveSettings(), and SetSaves().

8.19.4 Property Documentation

8.19.4.1 Instance

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

Definition at line 10 of file StorageManager.cs.

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

Referenced by MainMenu._Ready(), _Ready(), MainMenu.Change(), MainMenu.OnContinueButtonPressed(), MainMenu.OnDeleteConfirmationConfirmed(), MainMenu.OnQuitButtonPressed(), MainMenu.OnSave1SelectPressed(), MainMenu.OnSave2SelectPressed(), MainMenu.OnSave3SelectPressed(), Hud.OnSaveButtonPressed(), Hud.OnSaveMenuButtonPressed(), and Hud.OnSaveQuitButtonPressed().

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

- C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/StorageManager.cs

Chapter 9

File Documentation

9.1 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/Dock.cs File Reference

Namespaces

- namespace GdMUT
- namespace GdMUT.Components

9.2 Dock.cs

Go to the documentation of this file.

```
// Copyright (c) Spencer (Spycemyster) Chang, LLC. All Rights Reserved.
// Licensed under the MIT License. See LICENSE in the project root for license information.
namespace GdMUT.Components;

using Godot;
using System;
using System.Diagnostics;
using System.Threading;

#if TOOLS
[Tool]
public partial class Dock : Control
{
    private const string TEST_RESULT_SCENE = "res://addons/GDMUT/TestResult.tscn";
    private readonly System.Collections.Generic.Dictionary<
        Type,
        System.Collections.Generic.List<TestFunction>
    > _testDictionary = new();
    private readonly System.Collections.Generic.Dictionary<Type, TestResult> _testResultDictionary =
        new();

    [Export]
    private LineEdit _filter;

    [Export]
    private CheckBox _multithreadedEnabled;

    [Export]
    private LineEdit _numThreads;

    [Export]
    private Button _runTests;

    [Export]
    private Button _loadTests;

    [Export]
    private VBoxContainer _testList;

    private System.Collections.Generic.List<TestFunction> _tests = new();

    public override void _EnterTree()
    {
        base._EnterTree();
        _runTests.Pressed += RunTests;
        _loadTests.Pressed += LoadTests;
    }

    private void LoadTests()
    {
        var stopwatch = new Stopwatch();
        stopwatch.Start();
        foreach (Node node in _testList.GetChildren())
        {
            node.QueueFree();
        }

        _tests?.Clear();
        _tests = TestLoader.SearchForAllTests();
        _testDictionary.Clear();
        for (int testIndex = 0; testIndex < _tests.Count; testIndex++)
        {
            TestFunction function = _tests[testIndex];
            if (!function.Name.Contains(_filter.Text))
            {
                continue;
            }

            if (
                _testDictionary.TryGetValue(
                    function.Type,
                    out System.Collections.Generic.List<TestFunction> testList
                )
            )
            {
                testList.Add(function);
            }
            else
            {
                _testDictionary.Add(
                    function.Type,
                    new System.Collections.Generic.List<TestFunction>() { function }
                );
            }
        }

        _testResultDictionary.Clear();
        var testResultScene = GD.Load<PackedScene>(TEST_RESULT_SCENE);
        foreach (Type type in _testDictionary.Keys)
        {
            var functions = _testDictionary[type];
            var testResult = testResultScene.Instantiate<TestResult>();
            testResult.SetTypeName(type.Name);
            _testList.AddChild(testResult);
            _testResultDictionary.Add(type, testResult);
            foreach (TestFunction function in functions)
            {
                testResult.AddMethodResult(function);
            }
        }

        stopwatch.Stop();
        GD.Print($"Loading tests took {stopwatch.ElapsedMilliseconds}ms");
    }

    private void RunTestsInRange(int startIndex, int endIndex)
    {
        for (int testIndex = startIndex; testIndex < endIndex; testIndex++)
        {
            var test = _tests[testIndex];
            GD.Print(test.Name);
            Result testResult;
            try
            {
                testResult = (Result)test.Method.Invoke(null, null);
            }
            catch (Exception e)
            {
                testResult = new Result(false, $"Exception thrown: {e.Message}");
            }

            test.Result = testResult;
        }
    }

    private void RunTests()
    {
        if (_tests.Count == 0)
        {
            GD.Print("No tests loaded");
            return;
        }

        var stopwatch = new Stopwatch();
        stopwatch.Start();

        if (
            _multithreadedEnabled.ButtonPressed
            && int.TryParse(_numThreads.Text, out int numThreads)
            && numThreads > 0
        )
        {
            GD.Print("Run Tests multithreaded");
            Thread[] threads = new Thread[numThreads];
            int testsPerThread =
                (_tests.Count / numThreads) + (_tests.Count % numThreads > 0 ? 1 : 0);
            for (int threadIndex = 0; threadIndex < numThreads; threadIndex++)
            {
                int startIndex = threadIndex * testsPerThread;
                int endIndex = Math.Min((threadIndex + 1) * testsPerThread, _tests.Count);
                threads[threadIndex] = new Thread(() => RunTestsInRange(startIndex, endIndex));
                threads[threadIndex].Start();
            }

            foreach (Thread thread in threads)
            {
                thread.Join();
            }
        }
        else
        {
            GD.Print("Run Tests singlethreaded");
            RunTestsInRange(0, _tests.Count);
        }

        stopwatch.Stop();
        UpdateUIWithResults();
        GD.Print($"Tests took {stopwatch.ElapsedMilliseconds}ms");
    }

    private void UpdateUIWithResults()
    {
        foreach (TestResult result in _testResultDictionary.Values)
        {
            result.UpdateResult();
        }
    }
}
#endif
```

9.3 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/GDMUT.cs File Reference

9.4 GDMUT.cs

Go to the documentation of this file.

```
#if TOOLS
using Godot;

namespace GdMUT;

[Tool]
public partial class GDMUT : EditorPlugin
{
    private const string DOCK_SCENE = "res://addons/GDMUT/Dock.tscn";
    private Control _dock;

    public override void _EnterTree()
    {
        base._EnterTree();
        _dock = GD.Load<PackedScene>(DOCK_SCENE).Instantiate<Control>();
        AddControlToDock(DockSlot.RightUl, _dock);
        GD.Print("Successfully loaded GDMUT");
    }

    public override void _ExitTree()
    {
        base._ExitTree();
        RemoveControlFromDocks(_dock);
        _dock?.Free();
    }
}
#endif
```

9.5 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/MethodResult.cs File Reference

9.6 MethodResult.cs

Go to the documentation of this file.

```
#if TOOLS
using Godot;

namespace GdMUT.Components;

[Tool]
public partial class MethodResult : Control
{
    [Export]
    private RichTextLabel _methodName;

    [Export]
    private RichTextLabel _result;
    private TestFunction _function;

    public override void _EnterTree()
    {
        base._EnterTree();
    }

    public void SetMethodResult(TestFunction function)
    {
        _function = function;
        _methodName.Text = function.Method.Name;
        Reset();
    }

    public void Update()
    {
        SetSuccess(_function.Result.IsSuccess, _function.Result.Message);
    }

    public void Reset()
    {
        _result.Text = string.Empty;
        SelfModulate = new Color(1, 1, 1);
    }

    public void SetSuccess(bool isSuccess, string result = "")
    {
        _result.Text = (isSuccess ? "Success: " : "Failure: ") + result;
        Modulate = isSuccess ? new Color(0, 1, 0) : new Color(1, 0, 0);
        GD.Print($"{result} {isSuccess}");
    }
}
#endif
```

9.7 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/Result.cs File Reference

9.8 Result.cs

Go to the documentation of this file.

```
#if TOOLS
using System;

namespace GdMUT;

public struct Result
{
    public static readonly Result Success = new(true, string.Empty);

    public static readonly Result Failure = new(false, string.Empty);

    public Result(bool success, string message = "")
    {
        IsSuccess = success;
        Message = message;
    }

    public bool IsSuccess { get; set; }

    public string Message { get; set; }
}
#endif
```

9.9 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/TestFunction.cs File Reference

9.10 TestFunction.cs

Go to the documentation of this file.

```
#if TOOLS
using System;
using System.Reflection;

namespace GdMUT;

public class TestFunction
{
    public string Name { get; set; }

    public Type Type { get; set; }

    public MethodInfo Method { get; set; }

    public Result Result { get; set; }
}
#endif
```

9.11 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/addons/GDMUT/TestLoader.cs File Reference

9.12 TestLoader.cs

Go to the documentation of this file.

```
#if TOOLS
using System;
using System.Collections.Generic;
using System.Reflection;
using Godot;

namespace GdMUT;

public static class TestLoader
{
    public static List<TestFunction> SearchForAllTests()
    {
        List<TestFunction> tests = new();

        // get all functions with MonoTestFunctionAttribute
        ReadOnlySpan<Assembly> assemblies = AppDomain.CurrentDomain.GetAssemblies();
        for (int assemblyIndex = 0; assemblyIndex < assemblies.Length; assemblyIndex++)
        {
            Assembly assembly = assemblies[assemblyIndex];
            if (
                assembly.FullName.StartsWith("System.")
                || assembly.FullName.Equals("System")
                || assembly.FullName.StartsWith("Microsoft.")
                || assembly.FullName.StartsWith("GodotSharp")
                || assembly.FullName.StartsWith("GodotTools")
                || assembly.FullName.StartsWith("GodotPlugins")
                || assembly.FullName.StartsWith("JetBrains")
                || assembly.FullName.Equals("netstandard")
            )
            {
                continue;
            }

            GD.Print($"Loading tests from {assembly.FullName}");
            LoadFunctionsFromAssembly(tests, assembly);
        }

        return tests;
    }

    private static void LoadFunctionsFromAssembly(List<TestFunction> tests, Assembly assembly)
    {
        ReadOnlySpan<Type> types = assembly.GetTypes();
        for (int typeIndex = 0; typeIndex < types.Length; typeIndex++)
        {
            LoadFunctionsFromType(tests, types[typeIndex]);
        }
    }

    private static void LoadFunctionsFromType(List<TestFunction> tests, Type type)
    {
        ReadOnlySpan<MethodInfo> methods = type.GetMethods();
        foreach (var method in methods)
        {
            var attribute = method.GetCustomAttributes(typeof(CSTestFunctionAttribute), false);

            if (attribute.Length > 0)
            {
                if (method.ReturnType != typeof(Result))
                {
                    GD.PushError(
                        $"Method {method.Name} in {method.DeclaringType} does not return Result.
Skipping it..."
                    );
                    continue;
                }
                else if (!method.IsStatic)
                {
                    GD.PushError(
                        $"Method {method.Name} in {method.DeclaringType} is not static. Skipping
it..."
                    );
                    continue;
                }

                tests.Add(
                    new TestFunction()
                    {
                        Name = method.Name,
                        Type = method.DeclaringType,
                        Method = method,
                    }
                );
            }
        }
    }
}
```

```
[AttributeUsage(AttributeTargets.Method)]
public class CSTestFunctionAttribute : Attribute
{
    public CSTestFunctionAttribute() { }
}
#endif
```

9.13 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das← Spiel/anfaengerpraktikum/addons/GDMUT/TestResult.cs File Reference

9.14 TestResult.cs

Go to the documentation of this file.

```
#if TOOLS
using System.Collections.Generic;
using Godot;

namespace GdMUT.Components;

[Tool]
public partial class TestResult : Control
{
    private const string TYPE_NAME_FORMAT = "[b][font_size=24][center]{0}[/center][/font_size][/b]";
    private const string METHOD_RESULT_SCENE = "res://addons/GDMUT/MethodResult.tscn";

    [Export]
    private RichTextLabel _typeName;

    [Export]
    private VBoxContainer _methodList;

    private List<(MethodResult, TestFunction)> _functions = new();
    private string _typeNameStr;

    public override void _EnterTree()
    {
        base._EnterTree();
        foreach (Node child in _methodList.GetChildren())
        {
            child.QueueFree();
        }

        _functions.Clear();
    }

    public void SetTypeName(string typeName)
    {
        _typeNameStr = typeName;
        _typeName.Text = string.Format(TYPE_NAME_FORMAT, typeName);
    }

    public void UpdateResult()
    {
        int numSuccess = 0;
        foreach (var (methodResult, function) in _functions)
        {
            methodResult.Update();
            numSuccess += function.Result.IsSuccess ? 1 : 0;
        }

        _typeName.Text = string.Format(
            TYPE_NAME_FORMAT,
            _typeNameStr + $" ({numSuccess}/{_functions.Count})"
        );
        if (numSuccess == _functions.Count)
        {
            _typeName.Modulate = new Color(0, 1, 0);
        }
        else if (numSuccess == 0)
        {
            _typeName.Modulate = new Color(1, 0, 0);
        }
        else
        {
            _typeName.Modulate = new Color(1, 0.9f, 0);
        }
    }

    public void AddMethodResult(TestFunction function)
    {
        var methodResultScene = GD.Load<PackedScene>(METHOD_RESULT_SCENE);
        var methodResult = methodResultScene.Instantiate<MethodResult>();
        methodResult.SetMethodResult(function);
        _methodList.AddChild(methodResult);
        _functions.Add((methodResult, function));
    }
}
#endif
```

9.15 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩Spiel/anfaengerpraktikum/addons/godot-git-plugin/↩THIRDPARTY.md File Reference

9.16 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩Spiel/anfaengerpraktikum/README.md File Reference

9.17 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩Spiel/anfaengerpraktikum/scripts/BaseEnemy.cs File Reference

Classes

- class BaseEnemy

  Klasse für einen einfachen Gegner.

9.18 BaseEnemy.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class BaseEnemy : CharacterBody2D
{

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

    //customizable variables
    [Export]
    protected float Damage = 20f;
    [Export]
    protected bool Dead = false;
    [Export]
    protected bool Respawnable = true;
    [Export]
    protected float MaxHealthPoints = 100f;
    [Export]
    protected float Armor = 20f; //MUSS ZISCHEN 0 UND 99 LIEGEN
    [Export]
    protected float MaxStamina = 1f;
    [Export]
    protected float Speed = 10;
    [Export]
    protected int SinAmount = 10;
    [Export]
    protected double ReturnToStartAfter = 5;
    [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")]
    public uint Id { get; set;} = 0;

    //private variables
    protected float CurrentHealthPoints;
    protected float CurrentStamina;
    protected double ReturnToStart;
    protected bool Pursuing = false;
    protected Node2D CurrentTarget = null;
    protected Vector2 TargetPosition = Vector2.Inf;
    protected Vector2 StartPosition;
    protected bool StartRotation = false;
    private State AnimationState = State.IDLE;
    protected bool AlreadyHit = false;

    //linked nodes
    protected AnimatedSprite2D Sprite;
    protected CollisionPolygon2D CollisionPolygon;
    protected Area2D SwordHitbox;
    protected CollisionShape2D MainCollision;
    protected RayCast2D FrontCollisionRayCast;
    protected RayCast2D LineOfSight;
    protected RayCast2D LeftFallProtection;
    protected RayCast2D RightFallProtection;
    protected TextureProgressBar HealthBar;
    protected Player Player;

    public override void _Ready()
    {
        Sprite = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
        CollisionPolygon = GetNode<CollisionPolygon2D>("detection/CollisionPolygon2D");
        SwordHitbox = GetNode<Area2D>("AnimatedSprite2D/SwordHitBox");
        MainCollision = GetNode<CollisionShape2D>("MainCollision");
        FrontCollisionRayCast = GetNode<RayCast2D>("FrontCollisionRayCast");
        LineOfSight = GetNode<RayCast2D>("LineOfSight");
        LeftFallProtection = GetNode<RayCast2D>("LeftFallProtection");
        RightFallProtection = GetNode<RayCast2D>("RightFallProtection");
        HealthBar = GetNode<TextureProgressBar>("HealthBar");
        Player = GetNode<Player>("../../Player");

        CurrentHealthPoints = MaxHealthPoints;
        CurrentStamina = MaxStamina;
        ReturnToStart = ReturnToStartAfter;
        StartPosition = Position;
        StartRotation = Sprite.FlipH;

        HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;
    }

    public override void _Process(double DeltaTime)
    {
        HandleMovement(DeltaTime);
        if(CurrentStamina < MaxStamina){
            CurrentStamina += (float) DeltaTime;
            Velocity = Velocity * 0.8f;
        }
        if (!IsOnFloor() && !Dead) {
            Velocity += GetGravity() * (float)DeltaTime;
        }
        UpdateAnimation();
        MoveAndSlide();
        CheckPlayerHit();
    }

    public void OnDetectionBodyEntered(Node2D body){
        if(CheckLineOfSight(body)){
            Pursuing = true;
            CurrentTarget = body;
        }
    }

    public void OnPursuingRadiusBodyExited(Node2D body){
        if(body == CurrentTarget){
            Pursuing = false;
            CurrentTarget = null;
        }
    }

    public void OnHitboxAreaEntered(Area2D area){
        Player Player1 = (Player) area.GetParent().GetParent();
        TakeDamage(Player1.GetDamage());
    }

    public void OnSwordHitBoxBodyEntered(Node2D body){
        if(Dead) return;
        Sprite.Play("attack");
    }

    private void HandleMovement(double DeltaTime){
        if(Dead) return;
        if((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying()){
            Velocity = Vector2.Zero;
            return;
        }
        if(Pursuing){
            AnimationState = State.WALK;
            TargetPosition = CurrentTarget.Position;
            if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
                AnimationState = State.IDLE;
                Velocity = Vector2.Zero;
                return;
            }
            ReturnToStart = ReturnToStartAfter;
        } else if(ReturnToStart >= 0){
            AnimationState = State.IDLE;
            ReturnToStart -= DeltaTime;
            TargetPosition = Vector2.Inf;
        } else if(!IsCloseTo(Position.X, StartPosition.X, 0.1f)){
            AnimationState = State.WALK;
            TargetPosition = StartPosition;
        }

        if(TargetPosition != Vector2.Inf){

            if(IsCloseTo(Position.X, TargetPosition.X, 0.1f)){
                AnimationState = State.IDLE;
                Velocity = Vector2.Zero;
                if(TargetPosition == StartPosition && Sprite.FlipH != StartRotation){
                    FlipRotation();
                }
                TargetPosition = Vector2.Inf;
                return;
            }

            if(TargetPosition.X > Position.X){
                SetRotation(true);
                if(!FrontCollisionRayCast.IsColliding()){
                    Vector2 velocity = Vector2.Zero;
                    velocity.X = Speed;
                    Velocity = velocity;
                }
            } else {
                SetRotation(false);
                if(!FrontCollisionRayCast.IsColliding()){
                    Vector2 velocity = Vector2.Zero;
                    velocity.X = -Speed;
                    Velocity = velocity;
                }
            }

            if((!RightFallProtection.IsColliding() && !Sprite.FlipH) ||
(!LeftFallProtection.IsColliding() && Sprite.FlipH)){
                Velocity = Vector2.Zero;
            }

        } else {
            Velocity = Vector2.Zero;
            AnimationState = State.IDLE;
        }
    }


    protected virtual void UpdateAnimation(){
        if(Dead) return;
        if(!((Sprite.Animation == "take_hit" || Sprite.Animation == "attack") && Sprite.IsPlaying())){
            switch(AnimationState){
                case State.IDLE:
                    Sprite.Play("idle");
                    break;

                case State.WALK:
                    Sprite.Play("walk");
                    break;

                case State.ATTACK:
                    Sprite.Play("attack");
                    break;

                case State.TAKE_HIT:
                    Sprite.Play("take_hit");
                    break;
            }
        }
        HealthBar.Value = 100f* CurrentHealthPoints/MaxHealthPoints;

    }

    private void TakeDamage(Damage DMG){
        if(Dead) {
            return;
        }
        CurrentHealthPoints -= DMG.GetPhysicalDMG() * (1 - Armor / 100.0f) + DMG.GetTrueDMG();
        Position += DMG.GetPushAmount();
        if(CurrentHealthPoints <= 0){
            Die();
        } else {
            Sprite.Play("take_hit");
            if(DMG.GetSource() == Player){
                Pursuing = true;
                CurrentTarget = Player;
            }
        }
    }

    public bool IsDead(){
        return Dead;
    }

    private void CheckPlayerHit(){
        if(Dead) return;
        if(Sprite.Animation != "attack"){
            AlreadyHit = false;
            if(Sprite.Animation == "take_hit" || CurrentStamina < MaxStamina) return;
            Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    Sprite.Play("attack");
                }
            }
            return;
        }
        if(AlreadyHit) return;
        if(Sprite.Frame >= 6){
            CurrentStamina = 0;
            Godot.Collections.Array<Node2D> Bodies = SwordHitbox.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    Player.TakeDamage(new Damage(Damage, 0f, Vector2.Zero, this));
                    AlreadyHit = true;
                    break;
                }
            }
        }

    }

    private void Die(){
        Dead = true;
        Velocity = Vector2.Zero;
        MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, true);

        Sprite.Play("death");
        HealthBar.SetVisible(false);
        Player.SetSinAmount(PlayerStats.Instance.GetSinAmount() + SinAmount);

    }

    public void Respawn()
    {
        Dead = false;
        CurrentHealthPoints = MaxHealthPoints;
        HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
        MainCollision.SetDeferred(CollisionShape2D.PropertyName.Disabled, false);
        HealthBar.SetVisible(true);
        Sprite.Play("idle");
    }

    private bool CheckLineOfSight(Node2D body){
        Vector2 offset = Vector2.Zero;
        offset.Y = -14;
        LineOfSight.TargetPosition = body.Position + offset - (Position + LineOfSight.Position);
        if(LineOfSight.IsColliding()){
            return LineOfSight.GetCollider() == body;
        }
        return true;
    }

    private void FlipRotation(){
        Sprite.FlipH = !Sprite.FlipH;
        CollisionPolygon.RotationDegrees = Math.Abs(CollisionPolygon.RotationDegrees -180);
        SwordHitbox.RotationDegrees = Math.Abs(SwordHitbox.RotationDegrees -180);
        FrontCollisionRayCast.RotationDegrees = Math.Abs(FrontCollisionRayCast.RotationDegrees - 180);
    }

    private void SetRotation(bool Rotation){
        Sprite.FlipH = Rotation ^ StartRotation; // XOR mit StartRotation
        if(Rotation){
            CollisionPolygon.RotationDegrees = 180;
            SwordHitbox.RotationDegrees = 180;
            FrontCollisionRayCast.RotationDegrees = 180;
        } else {
            CollisionPolygon.RotationDegrees = 0;
            SwordHitbox.RotationDegrees = 0;
            FrontCollisionRayCast.RotationDegrees = 0;
        }
    }

    private bool IsCloseTo(float Value1, float Value2, float Delta){
        return Value1 <= (Value2 + Delta) && Value1 >= (Value2 - Delta);
    }
}
```

9.19 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/BloodVial.cs File Reference

Classes

- class BloodVial

  Klasse für die Interaktion zum heilen.

9.20 BloodVial.cs

Go to the documentation of this file.

```
using GdMUT;
using Godot;
using System;

public partial class BloodVial : Label {

    public override void _Ready() {
        Text = PlayerStats.Instance.GetBVCurrentUses() + "";
    }

    public void UseBloodVial(){
        if(PlayerStats.Instance.GetBVCurrentUses() <= 0) return;
        PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVCurrentUses() - 1);
        Text = PlayerStats.Instance.GetBVCurrentUses() + "";
        PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() +
PlayerStats.Instance.GetBVHealAmount());
    }

    public void ResetUses(){
        PlayerStats.Instance.SetBVCurrentUses(PlayerStats.Instance.GetBVMaxUses());
        Text = PlayerStats.Instance.GetBVCurrentUses() + "";
    }

    public void AddMaxUses(int Amount){
        PlayerStats.Instance.SetBVMaxUses(PlayerStats.Instance.GetBVMaxUses() + Amount);
        ResetUses();
    }

    public void LevelHealAmount(){
        PlayerStats.Instance.SetBVHealAmount(PlayerStats.Instance.GetBVHealAmount() + 25);
    }
}
```

9.21 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das←Spiel/anfaengerpraktikum/scripts/Boss1.cs File Reference

Classes

- class Boss1

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

9.22 Boss1.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Boss1 : BaseEnemy{

    private bool EnemiesRevived = false;
    private float RegenCooldown = 5.0f; // Zeit, nach der Regeneration beginnt, wenn kein Schaden
genommen wurde
    private float RegenTimer = 0.0f; // Timer für die Zeit seit dem letzten Angriff
    private float RegenAmount = 10.0f; // Menge an Gesundheit, die pro Tick regeneriert wird


    public override void _Ready(){

        MaxHealthPoints = 400f;
        Damage = 50f;
        Armor = 30f;
        Speed = 10f;
        SinAmount = 100; // Bonuspunkte für Spieler beim Besiegen des Bosses

        base._Ready();

        CurrentHealthPoints = MaxHealthPoints;
        HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
    }

    public override void _Process(double DeltaTime){
        base._Process(DeltaTime);

        if (CurrentHealthPoints <= MaxHealthPoints / 2 && !EnemiesRevived){
            StartGlowing();
            ReviveEnemies();
            EnemiesRevived = true;
            Armor = 60f; // Rüstung erhöhen
        }

        HandleRegeneration(DeltaTime);
    }

    private void HandleRegeneration(double DeltaTime){
        if (CurrentHealthPoints < MaxHealthPoints){
            RegenTimer += (float)DeltaTime;

            if (RegenTimer >= RegenCooldown){
                CurrentHealthPoints = Math.Min(CurrentHealthPoints + RegenAmount, MaxHealthPoints);
                HealthBar.Value = 100f * CurrentHealthPoints / MaxHealthPoints;
                RegenTimer = 0.0f; // Timer zurücksetzen
            }
        }
    }

    private void StartGlowing(){
        // Ändere die Modulationsfarbe des Sprites, um ein Leuchten zu simulieren
        if (Sprite != null){
            ShowPopupMessage("AHHHH!!!");
            Sprite.Modulate = new Color(1.0f, 0.84f, 0.0f, 1.0f); // Ein goldliche Leuchteffekt
        }
    }

    private void ShowPopupMessage(string Message){
        Label popup = new Label();
        popup.Text = Message;
        popup.AddThemeColorOverride("font_color", new Color(1, 0, 0)); // Rot
        popup.Modulate = new Color(1, 1, 1, 0); // Start transparent
        popup.AutowrapMode = TextServer.AutowrapMode.Word;
        popup.SizeFlagsHorizontal = (Control.SizeFlags)(int)Control.SizeFlags.ExpandFill;
        popup.SizeFlagsVertical = (Control.SizeFlags)(int)Control.SizeFlags.ShrinkCenter;
        popup.HorizontalAlignment = HorizontalAlignment.Center;
        popup.VerticalAlignment = VerticalAlignment.Center;


        Vector2 bossGlobalPosition = GetGlobalTransformWithCanvas().Origin;
        popup.GlobalPosition = bossGlobalPosition + new Vector2(0, -100);

        CanvasLayer canvas = new CanvasLayer();
        AddChild(canvas);
        canvas.AddChild(popup);

        var tween = CreateTween();
        tween.TweenProperty(popup, "modulate:a", 1, 0.5f).From(0); // Einblenden
        tween.TweenProperty(popup, "modulate:a", 0, 0.5f).From(1).SetDelay(1.0f); // Ausblenden nach 1 Sekunde
        tween.Connect("finished", new Callable(popup, "queue_free"));
    }

    private void ReviveEnemies()
    {
        // Hole den Elternknoten (bossRoom)
        Node BossRoom = GetParent();

        // Iteriere durch alle Kinder von bossRoom
        foreach (Node Child in BossRoom.GetChildren()){
            if (Child is BaseEnemy BaseEnemy && BaseEnemy.IsDead()){
                BaseEnemy.Respawn();
            }
        }
    }
}
```

9.23 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das Spiel/anfaengerpraktikum/scripts/Checkpoint.cs File Reference

Classes

- class Checkpoint

9.24 Checkpoint.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Checkpoint : Node2D
{

    // Variable für Player
    private Player Player;

    /*
     * @brief Intitalisierung der Node Player
     */
    public override void _Ready()
    {
        // Zugriff auf Player Node
        Player = GetNode<Player>("../Player");
    }

    /*
     * @brief Diese Funktion wird aufgerufen, wenn der Player den Checkpoint betritt
     * @param body Der Körper, der den Checkpoint betritt
     */
    private void OnPlayerBodyEntered(Node body)
    {

        if (body is Player Player)
        {
            // Setzen des Spawnpoints
            PlayerStats PlayerStats = GetNode<PlayerStats>("/root/PlayerStats");
            PlayerStats.Instance.SetSpawnPoint(this.GlobalPosition);
            Player.MaxHeal();
            PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetMaxStamina());
            Player.GetBloodVials().ResetUses();
            GD.Print("Spawnpoint des Players gesetzt auf: ", this.GlobalPosition);

            PlayerStats.SetRespawnLevelTag(GetParent().Name);
            GD.Print("RespawnLevelTag des Players gesetzt auf: ", GetParent().Name);
            GD.Print(PlayerStats.Instance.GetRespawnLevelTag());
        }

    }
}
```

9.25 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/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.

9.26 Damage.cs

Go to the documentation of this file.

```
using Godot;

public class Damage{

    private float PhysicalDMG;
    private float TrueDMG;
    private Vector2 PushAmount;
    private Node2D Source;

    public Damage(float PhysicalDMG, float TrueDMG, Vector2 PushAmount, Node2D Source){
        this.PhysicalDMG = PhysicalDMG;
        this.TrueDMG = TrueDMG;
        this.PushAmount = PushAmount;
        this.Source = Source;
    }

    public float GetPhysicalDMG(){
        return PhysicalDMG;
    }

    public float GetTrueDMG(){
        return TrueDMG;
    }

    public Vector2 GetPushAmount(){
        return PushAmount;
    }

    public Node2D GetSource(){
        return Source;
    }
}
```

9.27 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/Door.cs File Reference

Classes

- class Door

  Klasse für die Tür.

9.28 Door.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Door : Area2D
{
    public Node Spawn;

    [Export]
    public string DestinationLevelTag { get; set; }

    [Export]
    public string DestinationDoorTag { get; set; }

    [Export]
    public string SpawnDirection { get; set; } = "up";



    public override void _Ready()
    {
        Spawn = GetNode("Spawn");
    }


    private void OnPlayerBodyEntered(Node body)
    {
        if (body is Player player)
        {
            var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
            NavigationManager.GoToLevel(DestinationLevelTag, DestinationDoorTag);
        }
    }
}
```

9.29 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/HealthBar.cs File Reference

Classes

- class HealthBar

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

9.30 HealthBar.cs

Go to the documentation of this file.

```
using Godot;

public partial class HealthBar : TextureProgressBar {

    public override void _Ready() {
        // Setze die maximale Gesundheit der HealthBar basierend auf der Spieler-MaxHealth
        MaxValue = PlayerStats.Instance.GetMaxHealthPoints();
        Value = PlayerStats.Instance.GetCurrentHealth();
    }

    public override void _Process(double DeltaTime) {
        // Aktualisiere den Wert der HealthBar basierend auf der aktuellen Gesundheit des Spielers
        Value = PlayerStats.Instance.GetCurrentHealth();
    }
}
```

9.31 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/Hud.cs File Reference

Classes

- class Hud

  Klasse für das PauseMenu.

9.32 Hud.cs

Go to the documentation of this file.

```
using Godot;
using System;


public partial class Hud : CanvasLayer {

    private AnimationPlayer AnimationPlayer;
    private CenterContainer Buttons;
    private bool Enabled;


    public override void _Ready() {
        AnimationPlayer = GetNode<AnimationPlayer>("PauseMenu/AnimationPlayer");
        Buttons = GetNode<CenterContainer>("PauseMenu/Buttons");
        AnimationPlayer.Play("RESET");
    }

    public override void _Process(double DeltaTime) {
        if(Input.IsActionJustPressed("escape")){
            TogglePause();
        }
    }

    private void TogglePause(){
        Enabled = !Enabled;
        GetTree().Paused = Enabled;
        if(Enabled){
            AnimationPlayer.Play("Pause");
            Buttons.Visible = true;
        } else {
            AnimationPlayer.PlayBackwards("Pause");
            Buttons.Visible = false;
        }
    }

    public void OnResumeButtonPressed(){
        TogglePause();
    }

    public void OnSaveButtonPressed(){
        StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
    }

    public void OnSaveMenuButtonPressed(){
        StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
        NavigationManager.Instance.GoToLevel("main_menu", null);
        PlayerStats.Instance.Reload();
        GetTree().Paused = false;
    }

    public void OnSaveQuitButtonPressed(){
        StorageManager.Instance.SaveAll(StorageManager.Instance.GetLastSaveId());
        GetTree().Quit();
    }

}
```

9.33 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/Interactable.cs File Reference

Classes

- class Interactable

Klasse für Interaktion.

9.34 Interactable.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Interactable : AnimatedSprite2D {

    private Player Player;
    private RichTextLabel TextLabel;
    private Control PopUp;
    private Area2D Area;

    [Export(PropertyHint.MultilineText)]
    private String Text { get; set;}

    public override void _Ready(){
        Player = GetNode<Player>("../Player");
        TextLabel = GetNode<RichTextLabel>("../HUD/PopUp/Text");
        PopUp = GetNode<Control>("../HUD/PopUp");
        Area = GetNode<Area2D>("Area2D");
    }

    public override void _Process(double DeltaTime){
        if(Input.IsActionJustPressed("interact")){
            Godot.Collections.Array<Node2D> Bodies = Area.GetOverlappingBodies();
            foreach(Node2D Body in Bodies){
                if(Body == Player){
                    TextLabel.Clear();
                    TextLabel.AppendText(Text);
                    PopUp.Visible = true;
                    return;
                }
            }
        }
    }

    public void OnAreaBodyExited(Node2D Body){
        if(Body == Player){
            PopUp.Visible = false;
            TextLabel.Clear();
        }
    }

}
```

9.35 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/LevelManager.cs File Reference

Classes

- class LevelManager

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

9.36 LevelManager.cs

Go to the documentation of this file.

```
using Godot;

public partial class LevelManager : Node2D
{
    public override void _Ready()
    {
        var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");

        if (NavigationManager.SpawnDoorTag != null)
        {
            OnLevelSpawn(NavigationManager.SpawnDoorTag);
        }
        else
        {
            NavigationManager.CallDeferred("TriggerPlayerSpawn", PlayerStats.Instance.GetPosition(), "");
        }

    }

    private void OnLevelSpawn(string DestinationTag)
    {
        var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
        // Pfad zur Tür basierend auf dem Ziel-Tag erstellen
        string DoorPath = "Doors/Door_" + DestinationTag;

        Door door = GetNode<Door>(DoorPath);

        // TriggerPlayerSpawn nach deferred ausführen
        NavigationManager.CallDeferred("TriggerPlayerSpawn", door.GlobalPosition, door.SpawnDirection);
    }
}
```

9.37 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/MainMenu.cs File Reference

Classes

- class MainMenu

    Klasse für das MainMenu.

9.38 MainMenu.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class MainMenu : Node2D {

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


    public override void _Ready() {
        Navigation = GetNode<VBoxContainer>("Control/Navigation");
        SavesContainer = GetNode<MarginContainer>("Control/Saves");
        ContinueButton = GetNode<Button>("Control/Navigation/ContinueButton");
        InfoLabel = GetNode<Label>("Control/Saves/VBoxContainer/Info");

        SaveLabel[0] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Label");
        SelectButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Select");
        DeleteButton[0] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save1/Delete");
        SaveLabel[1] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Label");
        SelectButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Select");
        DeleteButton[1] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save2/Delete");
        SaveLabel[2] = GetNode<Label>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Label");
        SelectButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Select");
        DeleteButton[2] = GetNode<Button>("Control/Saves/VBoxContainer/HBoxContainer/Save3/Delete");

        DeleteConfirmation = GetNode<ConfirmationDialog>("DeleteConfirmation");

        if(StorageManager.Instance.GetLastSaveId() > -1){
            ContinueButton.Visible = true;
        }
    }


    private void Change(){
        if(MenuState == 0){
            SavesContainer.Visible = false;
            Navigation.Visible = true;
        } else {
            Navigation.Visible = false;
            SavesContainer.Visible = true;

            int Saves = StorageManager.Instance.GetSaves();

            if(MenuState == 1){
                InfoLabel.Text = "Select empty save to start a new Game";
                for(int i = 0; i < 3; i++){
                    if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
                        SaveLabel[i].Text = "Save " + (i+1);
                        SelectButton[i].Disabled = true;
                        DeleteButton[i].Disabled = false;
                    } else {
                        SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
                        SelectButton[i].Disabled = false;
                        DeleteButton[i].Disabled = true;
                    }
                }
            } else {
                InfoLabel.Text = "Select save to load Game";
                for(int i = 0; i < 3; i++){
                    if((Saves & (int) Math.Pow(2, i)) == (int) Math.Pow(2, i)){
                        SaveLabel[i].Text = "Save " + (i+1);
                        SelectButton[i].Disabled = false;
                        DeleteButton[i].Disabled = false;
                    } else {
                        SaveLabel[i].Text = "Save " + (i+1) + "\nEmpty";
                        SelectButton[i].Disabled = true;
                        DeleteButton[i].Disabled = true;
                    }
                }
            }
        }
    }

    public void OnContinueButtonPressed(){
        StorageManager.Instance.LoadGameFile(StorageManager.Instance.GetLastSaveId());
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
    }

    public void OnQuitButtonPressed(){
        StorageManager.Instance.SaveSettings();
        GetTree().Quit();
    }

    public void OnNewGameButtonPressed(){
        MenuState = 1;
        Change();
    }

    public void OnLoadGameButtonPressed(){
        MenuState = 2;
        Change();
    }

    public void OnBackButtonPressed(){
        MenuState = 0;
        Change();
    }

    public void OnSave1SelectPressed(){
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(0);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 1);
        StorageManager.Instance.SetLastSaveId(0);
    }

    public void OnSave1DeletePressed(){
        SaveToDelete = 1;
        DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
        DeleteConfirmation.Show();
    }

    public void OnSave2SelectPressed(){
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(1);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 2);
        StorageManager.Instance.SetLastSaveId(1);
    }

    public void OnSave2DeletePressed(){
        SaveToDelete = 2;
        DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
        DeleteConfirmation.Show();
    }

    public void OnSave3SelectPressed(){
        if(MenuState == 2){
            StorageManager.Instance.LoadGameFile(2);
        }
        NavigationManager.Instance.GoToLevel(PlayerStats.Instance.GetCurrentLevelTag(), null);
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() | 4);
        StorageManager.Instance.SetLastSaveId(2);
    }

    public void OnSave3DeletePressed(){
        SaveToDelete = 3;
        DeleteConfirmation.SetText("Are you sure you want to DELETE Save " + SaveToDelete + "?");
        DeleteConfirmation.Show();
    }

    public void OnDeleteConfirmationCanceled(){
        SaveToDelete = 0;
        Change();
    }

    public void OnDeleteConfirmationConfirmed(){
        StorageManager.Instance.SetSaves(StorageManager.Instance.GetSaves() ^ (int) Math.Pow(2,
  SaveToDelete - 1));
        Change();
    }

    public void OnDeleteConfirmationCloseRequested(){
        OnDeleteConfirmationCanceled();
    }

}
```

9.39 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/MainMenuBackground.cs File Reference

Classes

- class MainMenuBackground

  Klasse für die MainMenuBackground-Animation.

9.40 MainMenuBackground.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class MainMenuBackground : ParallaxLayer {

    [Export]
    private float ScrollSpeed = -10f;

    public override void _Process(double DeltaTime) {
        float X = GetMotionOffset().X;
        X += ScrollSpeed * (float) DeltaTime;
        SetMotionOffset(new Vector2(X,0));
    }
}
```

9.41 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩Spiel/anfaengerpraktikum/scripts/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.

9.42 NavigationManager.cs

Go to the documentation of this file.

```
using Godot;

public partial class NavigationManager : Node
{
    public static NavigationManager Instance { get; private set; }
    // Deklarieren der vorab geladenen Szenen
    private static readonly PackedScene SceneMainMenu =
(PackedScene)GD.Load("res://Scenes/main_menu.tscn");
    private static readonly PackedScene SceneIntro = (PackedScene)GD.Load("res://Scenes/intro.tscn");
    private static readonly PackedScene SceneLevel1 =
(PackedScene)GD.Load("res://Scenes/level1.tscn");
    private static readonly PackedScene SceneBoss =
(PackedScene)GD.Load("res://Scenes/bossRoom.tscn");
    private static readonly PackedScene SceneLevelOne =
(PackedScene)GD.Load("res://Scenes/level_one.tscn");
    private static readonly PackedScene SceneLevelTwo =
(PackedScene)GD.Load("res://Scenes/level_two.tscn");

    // Die Spawn-Tag-Variable
    public string SpawnDoorTag { get; private set; }

    [Signal]
    public delegate void OnTriggerPlayerSpawnEventHandler(Vector2 Position, string Direction);

    public override void _Ready(){
        Instance = this;
    }

    public void GoToLevel(string LevelTag, string DestinationTag)
    {
        PackedScene SceneToLoad = null;

        // Bestimmen, welches Level geladen werden soll
        switch (LevelTag)
        {
            case "main_menu":
                SceneToLoad = SceneMainMenu;
                break;
            case "intro":
                SceneToLoad = SceneIntro;
                break;
            case "level1":
                SceneToLoad = SceneLevel1;
                break;
            case "bossRoom":
                SceneToLoad = SceneBoss;
                break;
            case "level_one":
                SceneToLoad = SceneLevelOne;
                break;
            case "level_two":
                SceneToLoad = SceneLevelTwo;
                break;
        }

        // Überprüfen, ob eine Szene ausgewählt wurde und diese dann laden
        if (SceneToLoad != null){
            if(SceneToLoad != SceneMainMenu){
                PlayerStats.Instance.SetCurrentLevelTag(LevelTag);
                SpawnDoorTag = DestinationTag;
            }
            // Verwendung der ChangeSceneToPacked-Methode in Godot 4
            CallDeferred(nameof(DeferredChangeScene), SceneToLoad);
        }
    }

    private void DeferredChangeScene(PackedScene SceneToLoad)
    {
        GetTree().ChangeSceneToPacked(SceneToLoad);
    }

    public void TriggerPlayerSpawn(Vector2 Position, string Direction)
    {
        EmitSignal(SignalName.OnTriggerPlayerSpawn, Position, Direction);
    }
}
```

9.43 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/Player.cs File Reference

Classes

- class Player

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

9.44 Player.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Player : CharacterBody2D
{
    // Variablen für Bewegung, Sprünge und Dash
    private const float SPEED = 100f;
    private const float JUMP_VELOCITY = -300f;
    private int JumpMax = 2;
    private int JumpCount = 0;

    private Vector2 DashDirection = Vector2.Zero;
    private float DashSpeed = 300f;
    private bool IsDashing = false;
    private bool CanDash = true;
    private float DashTrailInterval = 0.05f;
    private float DashTrailTimer = 0f;

    // Referenzen zu den Knoten
    private AnimationPlayer AnimationPlayer;
    private Sprite2D Sprite;
    private Timer DashEffect;
    private Timer DashTimer;
    private CollisionShape2D SwordCollision;
    private CollisionShape2D PlayerHitbox;
    private BloodVial BloodVials;
    private Label SinDisplay;

    private Vector2 HauptHitbox;
    private int LastAttack = 0;

    //Variablen für Stamina
    private float TimeSinceLastStaminaUse = 0f;

    public override void _Ready() {
        AnimationPlayer = GetNode<AnimationPlayer>("AnimationPlayer");
        Sprite = GetNode<Sprite2D>("Sprite2D");
        DashEffect = GetNode<Timer>("DashEffect");
        DashTimer = GetNode<Timer>("DashTimer");
        SwordCollision = GetNode<CollisionShape2D>("Sprite2D/SwordHit/SwordCollision");
        PlayerHitbox = GetNode<CollisionShape2D>("PlayerHitbox");
        HauptHitbox = PlayerHitbox.Position;
        BloodVials = GetNode<BloodVial>("../HUD/BloodVial/Counter");
        SinDisplay = GetNode<Label>("../HUD/SinAmount/Counter");

        SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";

        NavigationManager navigationManager = GetNode<NavigationManager>("/root/NavigationManager");
        navigationManager.Connect("OnTriggerPlayerSpawn", new Callable(this, nameof(OnSpawn)));

        Position = PlayerStats.Instance.GetPosition();
    }

    public override void _PhysicsProcess(double DeltaTime) {
        // Gravitation hinzufügen, wenn der Charakter nicht am Boden ist
        if (!IsOnFloor()) {
            Velocity += GetGravity() * (float)DeltaTime;
        } else {
            CanDash = true; // Dash wird zurückgesetzt, wenn der Charakter am Boden ist
        }

        TimeSinceLastStaminaUse += (float)DeltaTime;
        RegenerateStamina(20f, DeltaTime);

        // Heal
        if(Input.IsActionJustPressed("heal")){
            BloodVials.UseBloodVial();
        }

        HandleJump();
        HandleMovement(DeltaTime);
        MoveAndSlide();
        UpdateAnimations();
        PlayerStats.Instance.SetPosition(Position);
        }

    private void HandleJump() {
        // Sprungzähler zurücksetzen, wenn der Charakter am Boden ist
        if (JumpCount != 0 && IsOnFloor()) {
            JumpCount = 0;
        }

        // Überprüfen, ob der Sprung-Button gedrückt wurde und der Charakter noch Sprünge übrig hat
        if (Input.IsActionJustPressed("ui_up") && JumpCount < JumpMax) {
            if (JumpCount == 0) {
            // Erster Sprung ohne Stamina-Verlust
            Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
            JumpCount += 1;
            } else if (JumpCount > 0) {
                // Beim Doppelsprung Stamina prüfen und abziehen
                if (UseStamina(15)) {
                    Velocity = new Vector2(Velocity.X, JUMP_VELOCITY);
                    JumpCount += 1;
                }
            }
        }
    }

    private void HandleMovement(double DeltaTime) {
        Vector2 direction = new Vector2(Input.GetAxis("ui_left", "ui_right"), Input.GetAxis("ui_up",
"ui_down")).Normalized();
        float currentSpeed = SPEED;

        // Sprite umdrehen basierend auf der Bewegungsrichtung und Kollision umdrehen
        if (direction.X < 0) {
            Sprite.FlipH = true;
            SwordCollision.Position = new Vector2(-Mathf.Abs(SwordCollision.Position.X),
SwordCollision.Position.Y);
            PlayerHitbox.Position = new Vector2(Sprite.Position.X * 1.8f, PlayerHitbox.Position.Y);
        } else if (direction.X > 0) {
            Sprite.FlipH = false;
            SwordCollision.Position = new Vector2(Mathf.Abs(SwordCollision.Position.X),
SwordCollision.Position.Y);
            PlayerHitbox.Position = HauptHitbox;
        }

        // Geschwindigkeit reduzieren, wenn der Spieler angreift
        if (AnimationPlayer.CurrentAnimation == "light_attack") {
            currentSpeed *= 0.5f;
        } else if (AnimationPlayer.CurrentAnimation == "heavy_attack") {
            currentSpeed *= 0.15f;
        }

        // Blockieren stoppt die Bewegung
        if (IsBlocking()) {
            currentSpeed = 0;
        }

        if (IsDashing) {
            DashInProgress(DeltaTime);
        } else {
            // Normale Bewegung verarbeiten, wenn kein Dash aktiv ist
            if (direction != Vector2.Zero) {
                Velocity = new Vector2(direction.X * currentSpeed, Velocity.Y);
            } else {
                Velocity = new Vector2(Mathf.MoveToward(Velocity.X, 0, SPEED), Velocity.Y);
            }

            // Überprüfen, ob der Dash-Button gedrückt wurde mit eine Bewegungsrichtung und nicht schon am angreifen ist
            if (Input.IsActionJustPressed("dash") && direction != Vector2.Zero && CanDash && !IsAttacking()) {
                // Wenn der Player genug Stamina hat kann er dashen
                if (UseStamina(20)){
                    DashDirection = direction;
                    StartDash();
                }
            }
        }
    }

    private void StartDash() {
        SetCollisionLayerValue(1,false);
        SetCollisionMaskValue(1,false);
        IsDashing = true;
        CanDash = false;
        DashTimer.Timeout += StopDash;
        DashTimer.Start();
        DashEffect.Start();
        DashTrailTimer = 0f;
    }

    private void DashInProgress(double DeltaTime) {
        // Charakter bewegt sich in die Dash-Richtung mit Dash-Geschwindigkeit
        if (DashDirection == Vector2.Up) {
            Velocity = DashDirection / 1.5f * DashSpeed;
        } else {
            Velocity = DashDirection * DashSpeed;
        }

        // Dash-Trail bei Intervallen erstellen
        DashTrailTimer -= (float)DeltaTime;
        if (DashTrailTimer <= 0f) {
            CreateDashEffect();
            DashTrailTimer = DashTrailInterval;
        }
    }

    private void CreateDashEffect() {
        Sprite2D PlayerCopyNode = (Sprite2D)Sprite.Duplicate();
        GetParent().AddChild(PlayerCopyNode);

        CollisionShape2D SwordCollisionCopy = PlayerCopyNode.GetNode<CollisionShape2D>("SwordHit/SwordCollision");
        if (SwordCollisionCopy != null) {
            SwordCollisionCopy.Disabled = true; // Deaktiviere die Kollision der Kopie
        }

        PlayerCopyNode.GlobalPosition = GlobalPosition + new Vector2(0, Sprite.Texture.GetHeight() * Sprite.Scale.Y * -0.5f);

        // Verblassen-Effekt für den Dash-Trail hinzufügen
        float AnimationTime = (float)(DashTimer.WaitTime / 3);

        Timer FadeTimer1 = new Timer();
        AddChild(FadeTimer1);
        FadeTimer1.Timeout += () => {
            if (IsInstanceValid(PlayerCopyNode)) {
                PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.4f);
            }
        };
        FadeTimer1.Start(AnimationTime);

        Timer FadeTimer2 = new Timer();
        AddChild(FadeTimer2);
        FadeTimer2.Timeout += () => {
            if (IsInstanceValid(PlayerCopyNode)) {
                PlayerCopyNode.Modulate = new Color(PlayerCopyNode.Modulate, 0.2f);
            }
        };
        FadeTimer2.Start(AnimationTime * 2);

        Timer FadeTimer3 = new Timer();
        AddChild(FadeTimer3);
        FadeTimer3.Timeout += () => {
            if (IsInstanceValid(PlayerCopyNode)) {
                PlayerCopyNode.QueueFree();
            }
        };
        FadeTimer3.Start(AnimationTime * 3);
    }

    private void StopDash() {
        IsDashing = false;
        DashEffect.Stop();
        DashTimer.Stop();
        DashTimer.Timeout -= StopDash;
        SetCollisionLayerValue(1,true);
        SetCollisionMaskValue(1,true);
    }

    private bool IsAttacking() {
        return AnimationPlayer.CurrentAnimation == "heavy_attack" || AnimationPlayer.CurrentAnimation == "light_attack";
    }

    private bool IsBlocking() {
        return AnimationPlayer.CurrentAnimation == "block";
    }

    public void MaxHeal(){
        PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetMaxHealthPoints());
    }

    public void TakeDamage(Damage Damage){
        float totalDamage = Damage.GetTrueDMG();
        if(!IsBlocking()){
            totalDamage += Damage.GetPhysicalDMG();
        } else {
            float CurrentStamina = PlayerStats.Instance.GetStamina();
            CurrentStamina -= Damage.GetPhysicalDMG();
            if(CurrentStamina < 0){
                totalDamage -= CurrentStamina;
            }
            PlayerStats.Instance.SetStamina(CurrentStamina);
        }

        PlayerStats.Instance.SetCurrentHealth(PlayerStats.Instance.GetCurrentHealth() - totalDamage);
        Position += Damage.GetPushAmount();

        // Überprüfe, ob der Spieler gestorben ist
        if (PlayerStats.Instance.GetCurrentHealth() <= 0){
            GD.Print("Spieler ist gestorben!");
            Respawn();
        }
    }

    public Damage GetDamage(){
        if(LastAttack == 1){
            return new Damage(50, 0, Vector2.Zero, this);
        }
        if(LastAttack == 2){
            Vector2 Push = new Vector2(20,0);
            if(Sprite.FlipH){
                Push = -Push;
            }
            return new Damage(100, 0, Push, this);
        }
        return new Damage(0,0,Vector2.Zero, this);
    }

    public void RegenerateStamina(float Amount, double delta) {
        // Wenn die Verzögerungszeit erreicht wurde, regeneriere Stamina
        if (TimeSinceLastStaminaUse >= 1f) {
            PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() + Amount * (float)delta); // Regeneriere Stamina abhängig von der Zeit
        }
    }

    public bool UseStamina(float Amount) {
        // Versucht, eine bestimmte Menge an Stamina zu verbrauchen.
        // Gibt true zurück, wenn genug Stamina verfügbar war; andernfalls false.
        if (PlayerStats.Instance.GetStamina() >= Amount) {
            PlayerStats.Instance.SetStamina(PlayerStats.Instance.GetStamina() - Amount);
            TimeSinceLastStaminaUse = 0f;
            return true;
        }

        return false;
    }

    public void SlowPlayer(float SlowAmount){
        Velocity = new Vector2(Velocity.X * SlowAmount, Velocity.Y);
    }

    public void Respawn(){
        var NavigationManager = GetNode<NavigationManager>("/root/NavigationManager");
        NavigationManager.GoToLevel(PlayerStats.Instance.GetRespawnLevelTag(), "spawn");
        BloodVials.ResetUses();

    }

    public BloodVial GetBloodVials(){
        return BloodVials;
    }

    public void SetSinAmount(int Value) {
        // SinAmount muss immer >= 0 sein
        PlayerStats.Instance.SetSinAmount(Value);
        SinDisplay.Text = PlayerStats.Instance.GetSinAmount() + "";
    }

    private void OnSpawn(Vector2 position, string direction){

        // Spielerposition auf die übergebene Position setzen
        if (direction == "right")
        {
            // Update the x value by adding 50 to it, keep the original y value
            Sprite.FlipH = false;
            position = position with { X = position.X + 25 };
        }
        else if (direction == "left")
        {
            // Update the x value by subtracting 50 from it, keep the original y value
            Sprite.FlipH = true;
            position = position with { X = position.X - 25 };
        }
        Position = position;

    }


    private void UpdateAnimations() {
        if (Input.IsActionJustPressed("light_attack") && !IsDashing && !IsAttacking()) {
            if (UseStamina(10)){
                LastAttack = 1;
                AnimationPlayer.Play("light_attack");
            }
        } else if (Input.IsActionJustPressed("heavy_attack") && !IsDashing && !IsAttacking()) {
            if (UseStamina(25)){
                LastAttack = 2;
                AnimationPlayer.Play("heavy_attack");
            }
        }
        if (Input.IsActionPressed("block") && !IsDashing && !IsAttacking() && IsOnFloor()) {
            if (UseStamina(0)){
                AnimationPlayer.Play("block");
                LastAttack = 0;
            }
        }

        if (IsOnFloor() && !IsAttacking() && !IsBlocking()) {
            LastAttack = 0;
            if (Velocity.X == 0) {
                AnimationPlayer.Play("idle");
            } else {
                AnimationPlayer.Play("run");
            }
        } else if (!IsOnFloor() && !IsAttacking() && !IsBlocking()) {
            LastAttack = 0;
            if (Velocity.Y < 0) {
                AnimationPlayer.Play("jump");
            } else if (Velocity.Y > 0) {
                AnimationPlayer.Play("fall");
            }
        }
    }
}
```

9.45 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das←Spiel/anfaengerpraktikum/scripts/PlayerStats.cs File Reference

Classes

- class PlayerStats

  Klasse für die Spielerstats.

9.46 PlayerStats.cs

Go to the documentation of this file.

```
using System;
using Godot;

public partial class PlayerStats : Node
{

    public static PlayerStats Instance { get; private set; }

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


    public override void _Ready(){
        CurrentHealth = MaxHealthPoints;
        CurrentStamina = MaxStamina;
        BVCurrentUses = BVMaxUses;
        Instance = this;
    }

    public String GetRespawnLevelTag() {
        return RespawnLevelTag;
    }

    public void SetRespawnLevelTag(String levelTag) {
        RespawnLevelTag = levelTag;
    }

    public String GetCurrentLevelTag() {
        return CurrentLevelTag;
    }

    public void SetCurrentLevelTag(String levelTag) {
        CurrentLevelTag = levelTag;
    }

    public void SetSpawnPoint(Vector2 spawnPoint) {
        SpawnPoint = spawnPoint;
    }

    public Vector2 GetSpawnPoint(){
        return SpawnPoint;
    }

    public void SetPosition(Vector2 position) {
        Position = position;
    }

    public Vector2 GetPosition(){
        return Position;
    }


    public int GetSinAmount(){
        return SinAmount;
    }

    public void SetSinAmount(int Value) {
        // SinAmount muss immer >= 0 sein
        SinAmount = Mathf.Max(Value, 0);
    }

    public float GetMaxHealthPoints(){
        return MaxHealthPoints;
    }

    public void SetMaxHealthPoints(float maxHealthPoints){
        // MaxHealthPoints muss immer positiv sein
        MaxHealthPoints = Mathf.Max(maxHealthPoints, 1); // Verhindert, dass MaxHealthPoints <= 0 wird
    }

    public float GetCurrentHealth(){
        return CurrentHealth;
    }

    public void SetCurrentHealth(float Health){
        // CurrentHealth darf MaxHealthPoints nicht überschreiten.
        CurrentHealth = Mathf.Min(Health, MaxHealthPoints);
    }

    public void SetMaxStamina(float Value) {
        // MaxStamina muss immer positiv sein
        MaxStamina = Mathf.Max(Value, 1);
    }

    public float GetMaxStamina() {
        return MaxStamina;
    }

    public void SetStamina(float Value) {
        // Stellt sicher, dass die CurrentStamina im gültigen Bereich bleibt (zwischen 0 und
MaxStamina)
        CurrentStamina = Mathf.Clamp(Value, 0, MaxStamina);
    }

    public float GetStamina() {
        return CurrentStamina;
    }

    public void SetBVHealAmount(int Value){
        BVHealAmount = Math.Max(0, Value);
    }

    public int GetBVHealAmount() {
        return BVHealAmount;
    }

    public void SetBVMaxUses(int Value){
        BVMaxUses = Math.Max(0, Value);
    }

    public int GetBVMaxUses() {
        return BVMaxUses;
    }

    public void SetBVCurrentUses(int Value){
        BVCurrentUses = Math.Max(0, Value);
    }

    public int GetBVCurrentUses() {
        return BVCurrentUses;
    }

    public void Reload(){
        Instance = new PlayerStats();
        Instance._Ready();
    }

}
```

9.47 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/Spike.cs File Reference

Classes

- class Spike

  Klasse für die Spikes.

9.48 Spike.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class Spike : Node2D
```

```
{
    // Variable für Player
    private Player Player;


    [Export]
    private float Damage = 10f;

    public override void _Ready()
    {
        // Zugriff auf Player Node

        Player = GetNode<Player>("../../Player");
    }

    private void OnPlayerBodyEntered(Node body)
    {

        if (body is Player)
        {
            Player = (Player)body; // Instanzvariable setzen
            Player.TakeDamage(GetDamage());
            Player.SlowPlayer(0.5f);
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
            GD.Print("Player entered spike");
        }


    }

    private void OnPlayerBodyExited(Node body)
    {
        if (body is Player)
        {
            Player = null; // Instanzvariable zurücksetzen
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
        }
    }

    private void OnTimerTimeout()
    {
        GD.Print("Timer timeout");
        Player.TakeDamage(GetDamage());
        GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
    }

    public Damage GetDamage()
    {
        return new Damage(0, Damage, Vector2.Zero, this);
    }
}
```

9.49 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩ Spiel/anfaengerpraktikum/scripts/SpikeDynamic.cs File Reference

Classes

- class SpikeDynamic

  Klasse für die beweglichen Spikes.

9.50 SpikeDynamic.cs

Go to the documentation of this file.

```
using Godot;
using System;

public partial class SpikeDynamic : Node2D
{
    // Variable für Player
    private Player Player;

    [Export]
    private float Damage = 10f;

    public override void _Ready()
    {
        // Zugriff auf Player Node

        Player = GetNode<Player>("../../../Player");
    }

    private void OnPlayerBodyEntered(Node body)
    {

        if (body is Player)
        {
            Player = (Player)body; // Instanzvariable setzen
            Player.TakeDamage(GetDamage());
            Player.SlowPlayer(0.5f);
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
            GD.Print("Player entered spike");
        }


    }

    private void OnPlayerBodyExited(Node body)
    {
        if (body is Player)
        {
            Player = null; // Instanzvariable zurücksetzen
            GetNode<Timer>("StaticBody2D/Area2D/Timer").Stop();
        }
    }

    private void OnTimerTimeout()
    {
        GD.Print("Timer timeout");
        Player.TakeDamage(GetDamage());
        GetNode<Timer>("StaticBody2D/Area2D/Timer").Start();
    }

    public Damage GetDamage()
    {
        return new Damage(0, Damage, Vector2.Zero, this);
    }
}
```

9.51 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das↩Spiel/anfaengerpraktikum/scripts/StaminaBar.cs File Reference

Classes

- class StaminaBar

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

9.52 StaminaBar.cs

Go to the documentation of this file.

```
using Godot;

public partial class StaminaBar : TextureProgressBar {

    public override void _Ready() {
        // Setze die maximale Ausdauer der StaminaBar basierend auf der Spieler-MaxStamina
        MaxValue = PlayerStats.Instance.GetMaxStamina();
        Value = PlayerStats.Instance.GetStamina();
    }

    public override void _Process(double DeltaTime) {
        // Aktualisiere den Wert der StaminaBar basierend auf der aktuellen Ausdauer des Spielers
        Value = PlayerStats.Instance.GetStamina();
    }
}
```

9.53 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/Das←Spiel/anfaengerpraktikum/scripts/StorageManager.cs File Reference

Classes

- class StorageManager

  Klasse für das Speichern und Laden von Daten.

9.54 StorageManager.cs

Go to the documentation of this file.

```
using Godot;
using System;
using System.Collections;

public partial class StorageManager : Node {

    public static StorageManager Instance { get; private set; }
    private const String PathSettings = "user://settings.txt";
    private String[] PathSave = {"user://save1.dat", "user://save2.dat", "user://save3.dat"};
    private int LastSaveId = -1;
    private int Saves = 0;


    public override void _Ready(){
        LoadSettings();
        Instance = this;
    }

    public void LoadSettings(){
        if(!FileAccess.FileExists(PathSettings)){
            return;
        }
        FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Read);
        Saves = (int) File.GetVar();
        LastSaveId = (int) File.GetVar();

        File.Close();
    }

    public void LoadGameFile(int id){
        if(!FileAccess.FileExists(PathSave[id])){
            return;
        }
        FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Read);
        PlayerStats.Instance.SetRespawnLevelTag((String) File.GetVar());
        PlayerStats.Instance.SetCurrentLevelTag((String) File.GetVar());
        PlayerStats.Instance.SetSpawnPoint((Vector2) File.GetVar());
        PlayerStats.Instance.SetPosition((Vector2) File.GetVar());
        PlayerStats.Instance.SetSinAmount((int) File.GetVar());
        PlayerStats.Instance.SetMaxHealthPoints((float) File.GetVar());
        PlayerStats.Instance.SetCurrentHealth((float) File.GetVar());
        PlayerStats.Instance.SetMaxStamina((float) File.GetVar());
        PlayerStats.Instance.SetStamina((float) File.GetVar());
        PlayerStats.Instance.SetBVHealAmount((int) File.GetVar());
        PlayerStats.Instance.SetBVMaxUses((int) File.GetVar());
        PlayerStats.Instance.SetBVCurrentUses((int) File.GetVar());

        File.Close();
    }

    public void SaveAll(int id){
        SaveGameFile(id);
        SaveSettings();
    }

    public void SaveSettings(){
        FileAccess File = FileAccess.Open(PathSettings, FileAccess.ModeFlags.Write);
        File.StoreVar(Saves);
        File.StoreVar(LastSaveId);

        File.Close();
    }

    public void SaveGameFile(int id){
        FileAccess File = FileAccess.Open(PathSave[id], FileAccess.ModeFlags.Write);
        File.StoreVar(PlayerStats.Instance.GetRespawnLevelTag());
        File.StoreVar(PlayerStats.Instance.GetCurrentLevelTag());
        File.StoreVar(PlayerStats.Instance.GetSpawnPoint());
        File.StoreVar(PlayerStats.Instance.GetPosition());
        File.StoreVar(PlayerStats.Instance.GetSinAmount());
        File.StoreVar(PlayerStats.Instance.GetMaxHealthPoints());
        File.StoreVar(PlayerStats.Instance.GetCurrentHealth());
        File.StoreVar(PlayerStats.Instance.GetMaxStamina());
        File.StoreVar(PlayerStats.Instance.GetStamina());
        File.StoreVar(PlayerStats.Instance.GetBVHealAmount());
        File.StoreVar(PlayerStats.Instance.GetBVMaxUses());
        File.StoreVar(PlayerStats.Instance.GetBVCurrentUses());

        File.Close();
    }

    public void SetLastSaveId(int id){
        LastSaveId = id;
    }

    public int GetLastSaveId(){
        return LastSaveId;
    }

    public void SetSaves(int Saves){
        this.Saves = Saves;
    }

    public int GetSaves(){
        return Saves;
    }
}
```

9.55 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/TestClass.cs File Reference

Classes

- class **GdMUT.TestClass**

  This is a test class for GDMUT. This is purely for demonstration. If you added this into your project, feel free to delete it =).

Namespaces

- namespace GdMUT

9.56 C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/TestClass.cs

Go to the documentation of this file.

```
namespace GdMUT;

public static class TestClass
{
#if TOOLS
    [CSTestFunction]
    public static Result ExamplePass()
    {
        int x = 0;
        x *= 100;
        return (x == 0) ? Result.Success : Result.Failure;
    }

    [CSTestFunction]
    public static Result ExampleFail()
    {
        int x = 0;
        x *= 100;
        return (x != 0) ? Result.Success : Result.Failure;
    }

    [CSTestFunction]
    public static Result ExampleCustomFail()
    {
        int x = 0;
        x *= 100;
        return (x != 0)
            ? Result.Success
            : new Result(false, "You can't multiply 0 and expect anything else than 0!");
    }

    [CSTestFunction]
    public static Result ExampleCustomSuccess()
    {
        int x = 0;
        x *= 100;
        return (x == 0) ? new Result(true, "Proved that 0 * 100 = 0") : Result.Failure;
    }
#endif
}
```

Index

_PhysicsProcess
    Player, 88
_Process
    BaseEnemy, 33
    Boss1, 52
    HealthBar, 63
    Hud, 64
    Interactable, 68
    MainMenuBackground, 80
    StaminaBar, 120
_Ready
    BaseEnemy, 34
    BloodVial, 48
    Boss1, 52
    Checkpoint, 56
    Door, 61
    HealthBar, 63
    Hud, 65
    Interactable, 68
    LevelManager, 71
    MainMenu, 73
    NavigationManager, 82
    Player, 88
    PlayerStats, 103
    Spike, 114
    SpikeDynamic, 117
    StaminaBar, 120
    StorageManager, 122

AddMaxUses
    BloodVial, 48
AlreadyHit
    BaseEnemy, 42
anfaengerpraktikum, 1
AnimationPlayer
    Hud, 66
    Player, 97
AnimationState
    BaseEnemy, 42
Area
    Interactable, 69
Armor
    BaseEnemy, 43
ATTACK
    BaseEnemy, 33

BaseEnemy, 31
    _Process, 33
    _Ready, 34
    AlreadyHit, 42
    AnimationState, 42
    Armor, 43
    ATTACK, 33
    CheckLineOfSight, 34
    CheckPlayerHit, 35
    CollisionPolygon, 43
    CurrentHealthPoints, 43
    CurrentStamina, 43
    CurrentTarget, 43
    Damage, 43
    Dead, 44
    Die, 35
    FlipRotation, 36
    FrontCollisionRayCast, 44
    HandleMovement, 36
    HealthBar, 44
    Id, 47
    IDLE, 33
    IsCloseTo, 37
    IsDead, 39
    LeftFallProtection, 44
    LineOfSight, 44
    MainCollision, 44
    MaxHealthPoints, 45
    MaxStamina, 45
    OnDetectionBodyEntered, 39
    OnHitboxAreaEntered, 39
    OnPursuingRadiusBodyExited, 40
    OnSwordHitBoxBodyEntered, 40
    Player, 45
    Pursuing, 45
    Respawn, 40
    Respawnable, 45
    ReturnToStart, 45
    ReturnToStartAfter, 46
    RightFallProtection, 46
    SetRotation, 41
    SinAmount, 46
    Speed, 46
    Sprite, 46
    StartPosition, 46
    StartRotation, 47
    State, 33
    SwordHitbox, 47
    TAKE_HIT, 33
    TakeDamage, 41
    TargetPosition, 47
    UpdateAnimation, 42
    WALK, 33

BloodVial, 48
- _Ready, 48
- AddMaxUses, 48
- LevelHealAmount, 49
- ResetUses, 49
- UseBloodVial, 49

BloodVials
- Player, 97

Boss1, 50
- _Process, 52
- _Ready, 52
- EnemiesRevived, 54
- HandleRegeneration, 52
- RegenAmount, 54
- RegenCooldown, 55
- RegenTimer, 55
- ReviveEnemies, 53
- ShowPopupMessage, 53
- StartGlowing, 54

Buttons
- Hud, 66

BVCurrentUses
- PlayerStats, 111

BVHealAmount
- PlayerStats, 111

BVMaxUses
- PlayerStats, 111

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/Dock.cs, 127

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/GDMUT.cs, 129

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/MethodResult.cs, 130

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/Result.cs, 131

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/TestFunction.cs, 131

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/TestLoader.cs, 131

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/GDMUT/TestResult.cs, 133

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/addons/godot-git-plugin/THIRDPARTY.md, 134

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/README.md, 134

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BaseEnemy.cs, 134

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/BloodVial.cs, 138

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Boss1.cs, 139

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Checkpoint.cs, 140

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Damage.cs, 141

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/anfaengerpraktikum/scripts/Door.cs, 141, 142

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
142

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
143

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
143, 144

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
144

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
145

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
147

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
148

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
149

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
153, 154

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
155

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
156

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
157

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
158

C:/Users/Youssef/Desktop/UNI/S4/ComputerspielEntwicklung/DasSpiel/an
159

CanDash
- Player, 97

Change
- MainMenu, 73

CheckLineOfSight
- BaseEnemy, 34

CheckPlayerHit
- BaseEnemy, 35

Checkpoint, 55
- _Ready, 56
- OnPlayerBodyEntered, 56
- Player, 56

CollisionPolygon
- BaseEnemy, 43

ContinueButton
- MainMenu, 78

CreateDashEffect
- Player, 88

CurrentHealth
- PlayerStats, 112

CurrentHealthPoints
- BaseEnemy, 43

CurrentLevelTag
- PlayerStats, 112

CurrentStamina
- BaseEnemy, 43
- PlayerStats, 112

CurrentTarget
- BaseEnemy, 43

Damage, 57

BaseEnemy, 43
Damage, 57
GetPhysicalDMG, 58
GetPushAmount, 58
GetSource, 58
GetTrueDMG, 58
PhysicalDMG, 59
PushAmount, 59
Source, 59
Spike, 116
SpikeDynamic, 119
TrueDMG, 59
DashDirection
Player, 98
DashEffect
Player, 98
DashInProgress
Player, 89
DashSpeed
Player, 98
DashTimer
Player, 98
DashTrailInterval
Player, 98
DashTrailTimer
Player, 98
Dead
BaseEnemy, 44
DeferredChangeScene
NavigationManager, 82
DeleteButton
MainMenu, 78
DeleteConfirmation
MainMenu, 78
DestinationDoorTag
Door, 62
DestinationLevelTag
Door, 62
Die
BaseEnemy, 35
Door, 60
_Ready, 61
DestinationDoorTag, 62
DestinationLevelTag, 62
OnPlayerBodyEntered, 61
Spawn, 61
SpawnDirection, 62

Enabled
Hud, 67
EnemiesRevived
Boss1, 54

FlipRotation
BaseEnemy, 36
FrontCollisionRayCast
BaseEnemy, 44

GdMUT, 29
GdMUT.Components, 29
GetBloodVials
Player, 90
GetBVCurrentUses
PlayerStats, 103
GetBVHealAmount
PlayerStats, 103
GetBVMaxUses
PlayerStats, 103
GetCurrentHealth
PlayerStats, 104
GetCurrentLevelTag
PlayerStats, 104
GetDamage
Player, 90
Spike, 114
SpikeDynamic, 117
GetLastSaveId
StorageManager, 122
GetMaxHealthPoints
PlayerStats, 104
GetMaxStamina
PlayerStats, 105
GetPhysicalDMG
Damage, 58
GetPosition
PlayerStats, 105
GetPushAmount
Damage, 58
GetRespawnLevelTag
PlayerStats, 105
GetSaves
StorageManager, 122
GetSinAmount
PlayerStats, 106
GetSource
Damage, 58
GetSpawnPoint
PlayerStats, 106
GetStamina
PlayerStats, 106
GetTrueDMG
Damage, 58
GoToLevel
NavigationManager, 83

HandleJump
Player, 90
HandleMovement
BaseEnemy, 36
Player, 91
HandleRegeneration
Boss1, 52
HauptHitbox
Player, 99
HealthBar, 62
_Process, 63
_Ready, 63
BaseEnemy, 44

Hud, 64
  _Process, 64
  _Ready, 65
  AnimationPlayer, 66
  Buttons, 66
  Enabled, 67
  OnResumeButtonPressed, 65
  OnSaveButtonPressed, 65
  OnSaveMenuButtonPressed, 65
  OnSaveQuitButtonPressed, 66
  TogglePause, 66

Id
  BaseEnemy, 47
IDLE
  BaseEnemy, 33
InfoLabel
  MainMenu, 79
Instance
  NavigationManager, 85
  PlayerStats, 113
  StorageManager, 126
Interactable, 67
  _Process, 68
  _Ready, 68
  Area, 69
  OnAreaBodyExited, 68
  Player, 69
  PopUp, 69
  Text, 70
  TextLabel, 69
IsAttacking
  Player, 92
IsBlocking
  Player, 92
IsCloseTo
  BaseEnemy, 37
IsDashing
  Player, 99
IsDead
  BaseEnemy, 39

JUMP_VELOCITY
  Player, 99
JumpCount
  Player, 99
JumpMax
  Player, 99

LastAttack
  Player, 99
LastSaveId
  StorageManager, 125
LeftFallProtection
  BaseEnemy, 44
LevelHealAmount
  BloodVial, 49
LevelManager, 70
  _Ready, 71
  OnLevelSpawn, 71
LineOfSight
  BaseEnemy, 44
LoadGameFile
  StorageManager, 122
LoadSettings
  StorageManager, 123

MainCollision
  BaseEnemy, 44
MainMenu, 72
  _Ready, 73
  Change, 73
  ContinueButton, 78
  DeleteButton, 78
  DeleteConfirmation, 78
  InfoLabel, 79
  MenuState, 79
  Navigation, 79
  OnBackButtonPressed, 74
  OnContinueButtonPressed, 74
  OnDeleteConfirmationCanceled, 75
  OnDeleteConfirmationCloseRequested, 75
  OnDeleteConfirmationConfirmed, 75
  OnLoadGameButtonPressed, 75
  OnNewGameButtonPressed, 76
  OnQuitButtonPressed, 76
  OnSave1DeletePressed, 76
  OnSave1SelectPressed, 76
  OnSave2DeletePressed, 77
  OnSave2SelectPressed, 77
  OnSave3DeletePressed, 77
  OnSave3SelectPressed, 78
  SaveLabel, 79
  SavesContainer, 79
  SaveToDelete, 79
  SelectButton, 80
MainMenuBackground, 80
  _Process, 80
  ScrollSpeed, 81
MaxHeal
  Player, 92
MaxHealthPoints
  BaseEnemy, 45
  PlayerStats, 112
MaxStamina
  BaseEnemy, 45
  PlayerStats, 112
MenuState
  MainMenu, 79

Navigation
  MainMenu, 79
NavigationManager, 81
  _Ready, 82
  DeferredChangeScene, 82
  GoToLevel, 83
  Instance, 85
  OnTriggerPlayerSpawnEventHandler, 83

- SceneBoss, 84
- SceneIntro, 84
- SceneLevel1, 84
- SceneLevelOne, 85
- SceneLevelTwo, 85
- SceneMainMenu, 85
- SpawnDoorTag, 85
- TriggerPlayerSpawn, 84

OnAreaBodyExited
- Interactable, 68

OnBackButtonPressed
- MainMenu, 74

OnContinueButtonPressed
- MainMenu, 74

OnDeleteConfirmationCanceled
- MainMenu, 75

OnDeleteConfirmationCloseRequested
- MainMenu, 75

OnDeleteConfirmationConfirmed
- MainMenu, 75

OnDetectionBodyEntered
- BaseEnemy, 39

OnHitboxAreaEntered
- BaseEnemy, 39

OnLevelSpawn
- LevelManager, 71

OnLoadGameButtonPressed
- MainMenu, 75

OnNewGameButtonPressed
- MainMenu, 76

OnPlayerBodyEntered
- Checkpoint, 56
- Door, 61
- Spike, 115
- SpikeDynamic, 118

OnPlayerBodyExited
- Spike, 115
- SpikeDynamic, 118

OnPursuingRadiusBodyExited
- BaseEnemy, 40

OnQuitButtonPressed
- MainMenu, 76

OnResumeButtonPressed
- Hud, 65

OnSave1DeletePressed
- MainMenu, 76

OnSave1SelectPressed
- MainMenu, 76

OnSave2DeletePressed
- MainMenu, 77

OnSave2SelectPressed
- MainMenu, 77

OnSave3DeletePressed
- MainMenu, 77

OnSave3SelectPressed
- MainMenu, 78

OnSaveButtonPressed
- Hud, 65

OnSaveMenuButtonPressed
- Hud, 65

OnSaveQuitButtonPressed
- Hud, 66

OnSpawn
- Player, 93

OnSwordHitBoxBodyEntered
- BaseEnemy, 40

OnTimerTimeout
- Spike, 115
- SpikeDynamic, 118

OnTriggerPlayerSpawnEventHandler
- NavigationManager, 83

PathSave
- StorageManager, 125

PathSettings
- StorageManager, 126

PhysicalDMG
- Damage, 59

Player, 86
- _PhysicsProcess, 88
- _Ready, 88
- AnimationPlayer, 97
- BaseEnemy, 45
- BloodVials, 97
- CanDash, 97
- Checkpoint, 56
- CreateDashEffect, 88
- DashDirection, 98
- DashEffect, 98
- DashInProgress, 89
- DashSpeed, 98
- DashTimer, 98
- DashTrailInterval, 98
- DashTrailTimer, 98
- GetBloodVials, 90
- GetDamage, 90
- HandleJump, 90
- HandleMovement, 91
- HauptHitbox, 99
- Interactable, 69
- IsAttacking, 92
- IsBlocking, 92
- IsDashing, 99
- JUMP_VELOCITY, 99
- JumpCount, 99
- JumpMax, 99
- LastAttack, 99
- MaxHeal, 92
- OnSpawn, 93
- PlayerHitbox, 100
- RegenerateStamina, 93
- Respawn, 94
- SetSinAmount, 94
- SinDisplay, 100
- SlowPlayer, 94
- SPEED, 100
- Spike, 116

SpikeDynamic, 119
Sprite, 100
StartDash, 95
StopDash, 95
SwordCollision, 100
TakeDamage, 95
TimeSinceLastStaminaUse, 100
UpdateAnimations, 96
UseStamina, 97
PlayerHitbox
Player, 100
PlayerStats, 101
_Ready, 103
BVCurrentUses, 111
BVHealAmount, 111
BVMaxUses, 111
CurrentHealth, 112
CurrentLevelTag, 112
CurrentStamina, 112
GetBVCurrentUses, 103
GetBVHealAmount, 103
GetBVMaxUses, 103
GetCurrentHealth, 104
GetCurrentLevelTag, 104
GetMaxHealthPoints, 104
GetMaxStamina, 105
GetPosition, 105
GetRespawnLevelTag, 105
GetSinAmount, 106
GetSpawnPoint, 106
GetStamina, 106
Instance, 113
MaxHealthPoints, 112
MaxStamina, 112
Position, 112
Reload, 107
RespawnLevelTag, 113
SetBVCurrentUses, 107
SetBVHealAmount, 107
SetBVMaxUses, 108
SetCurrentHealth, 108
SetCurrentLevelTag, 108
SetMaxHealthPoints, 109
SetMaxStamina, 109
SetPosition, 109
SetRespawnLevelTag, 110
SetSinAmount, 110
SetSpawnPoint, 110
SetStamina, 111
SinAmount, 113
SpawnPoint, 113
PopUp
Interactable, 69
Position
PlayerStats, 112
Pursuing
BaseEnemy, 45
PushAmount
Damage, 59

RegenAmount
Boss1, 54
RegenCooldown
Boss1, 55
RegenerateStamina
Player, 93
RegenTimer
Boss1, 55
Reload
PlayerStats, 107
ResetUses
BloodVial, 49
Respawn
BaseEnemy, 40
Player, 94
Respawnable
BaseEnemy, 45
RespawnLevelTag
PlayerStats, 113
ReturnToStart
BaseEnemy, 45
ReturnToStartAfter
BaseEnemy, 46
ReviveEnemies
Boss1, 53
RightFallProtection
BaseEnemy, 46

SaveAll
StorageManager, 123
SaveGameFile
StorageManager, 124
SaveLabel
MainMenu, 79
Saves
StorageManager, 126
SavesContainer
MainMenu, 79
SaveSettings
StorageManager, 124
SaveToDelete
MainMenu, 79
SceneBoss
NavigationManager, 84
SceneIntro
NavigationManager, 84
SceneLevel1
NavigationManager, 84
SceneLevelOne
NavigationManager, 85
SceneLevelTwo
NavigationManager, 85
SceneMainMenu
NavigationManager, 85
ScrollSpeed
MainMenuBackground, 81
SelectButton

MainMenu, 80
SetBVCurrentUses
PlayerStats, 107
SetBVHealAmount
PlayerStats, 107
SetBVMaxUses
PlayerStats, 108
SetCurrentHealth
PlayerStats, 108
SetCurrentLevelTag
PlayerStats, 108
SetLastSaveId
StorageManager, 124
SetMaxHealthPoints
PlayerStats, 109
SetMaxStamina
PlayerStats, 109
SetPosition
PlayerStats, 109
SetRespawnLevelTag
PlayerStats, 110
SetRotation
BaseEnemy, 41
SetSaves
StorageManager, 125
SetSinAmount
Player, 94
PlayerStats, 110
SetSpawnPoint
PlayerStats, 110
SetStamina
PlayerStats, 111
ShowPopupMessage
Boss1, 53
SinAmount
BaseEnemy, 46
PlayerStats, 113
SinDisplay
Player, 100
SlowPlayer
Player, 94
Source
Damage, 59
Spawn
Door, 61
SpawnDirection
Door, 62
SpawnDoorTag
NavigationManager, 85
SpawnPoint
PlayerStats, 113
SPEED
Player, 100
Speed
BaseEnemy, 46
Spike, 114
_Ready, 114
Damage, 116
GetDamage, 114
OnPlayerBodyEntered, 115
OnPlayerBodyExited, 115
OnTimerTimeout, 115
Player, 116
SpikeDynamic, 116
_Ready, 117
Damage, 119
GetDamage, 117
OnPlayerBodyEntered, 118
OnPlayerBodyExited, 118
OnTimerTimeout, 118
Player, 119
Sprite
BaseEnemy, 46
Player, 100
StaminaBar, 119
_Process, 120
_Ready, 120
StartDash
Player, 95
StartGlowing
Boss1, 54
StartPosition
BaseEnemy, 46
StartRotation
BaseEnemy, 47
State
BaseEnemy, 33
StopDash
Player, 95
StorageManager, 121
_Ready, 122
GetLastSaveId, 122
GetSaves, 122
Instance, 126
LastSaveId, 125
LoadGameFile, 122
LoadSettings, 123
PathSave, 125
PathSettings, 126
SaveAll, 123
SaveGameFile, 124
Saves, 126
SaveSettings, 124
SetLastSaveId, 124
SetSaves, 125
SwordCollision
Player, 100
SwordHitbox
BaseEnemy, 47

TAKE_HIT
BaseEnemy, 33
TakeDamage
BaseEnemy, 41
Player, 95
TargetPosition
BaseEnemy, 47

Text
    Interactable, 70
TextLabel
    Interactable, 69
Third-Party Notices, 3
TimeSinceLastStaminaUse
    Player, 100
TogglePause
    Hud, 66
TriggerPlayerSpawn
    NavigationManager, 84
TrueDMG
    Damage, 59

UpdateAnimation
    BaseEnemy, 42
UpdateAnimations
    Player, 96
UseBloodVial
    BloodVial, 49
UseStamina
    Player, 97

WALK
    BaseEnemy, 33