TECNICAL DESIGN DOCUMENT

INTRO

This is the TDD "Technical Design Document" for the Project: Descent Remake.

Inside here you can find all the logic, workflow and guides of the scripts created for the game.

Check the Game Design Document for more infos about this project.

PHYSICS

All the scripts used when simulating the physics are using the in-builtphysics of Unity.

Unity in-built physics components used:

- RigidBody
- SphereCollider
- BoxCollider
- MeshCollider
- Raycast

These are the component used for the physics.

ARTIFICIAL INTELLIGENCE

- **LOGIC**

To achieve the desired behaviour written in the Game Design Document, the AI uses a Finite State Machine.

The State Machine has 3 states that acts as tasks:

- Chase
- Idle

- **SCRIPT'S LOGIC:**

The AI has two states.

```
IF the player IS close enough
    · DO Chase Player
Else
    · DO Idle
```

USER INTERFACE

- **WEAPONS**

To achieve the desired behaviour written in the Game Design Document, the interface IWeaponUI.cs is used to update in runtime the displayed name and icon of the weapons that are being used by the player.

- **MENU**

To achieve the desired behaviour written in the Game Design Document, each script related to the menu system uses UI buttons to change the screen settings, start the game, exit the game and change the volume settings.

SCORE SYSTEM

- **Logic**

To achieve the behavior written in the Game Design Document, the Singleton Pattern has been used to change the score parameter.

There are two ways to increase the score:

- Rescue the hostages;
- Kill the enemies.

The score will be reset if the player dies.

TIMER

To achieve the behavior written in the Game Design Document, the countdown timer is used to reload the scene after defeating the boss.

MINIMAP

To achieve the behavior written in the Game Design Document, the MinimapController.cs script disables the current scene and load the 3D minimap that can be seen by a rotateable and zoomable camera.

INPUTS

To achieve the behavior written in the Game Design Document, has been used the Unity Input System.

- **DEFAULT VALUES LIST**

- FORWARD = KeyCode.W;
- BACKWARD = KeyCode.S;
- TURNRIGHT = KeyCode.D;
- TURNLEFT = KeyCode.A;
- BANKRIGHT= KeyCode.Q;
- BANKLEFT= KeyCode.E;
- SLIDEUP = KeyCode.Space;
- SLIDEDOWN = KeyCode.Control;

- PITCHDOWN = YMousePosition;
- PITCHUP = YMousePosition;
- TURNRIGHT = XMousePosition;
- TURNLEFT = XMousePosition;
- PITCHDOWNRIGHT = XMousePosition + YMousePosition;
- PITCHDOWNLEFT = XMousePosition + YMousePosition;
- PITCHUPRIGHT = XMousePosition + YMousePosition;
- PITCHUPLEFT = XMousePosition + YMousePosition;

- LASERCANNON = KeyCode.1;
- VULCANGUN = KeyCode.2;
- CHANGESECONDARYWEAPON = MouseScroll;
- PRIMARYFIRE = MouseButton.1;
- SECONDARYFIRE = MouseButton.2;

- REARVIEW = KeyCode.R;

- ■ MENU= KeyCode.Esc;

ENTITY SYSTEM

To achieve the behavior written in the Game Design Document, the entity system uses interfaces to make enemies, doors and various objects vulnerables and destructables. To do so the system uses an IVulnerable interface that sets the HP of an object, and two base classes, one for destructable objects (like the enemies) and one for the objects that have phases of destruction (like the destructable vents).

PLAYER

- **PLAYER WEAPON SYSTEM**

To achieve the behavior written in the Game Design Document, the player uses the statemachine WeaponInventory.cs and WeaponState.cs to unlock and change the equipped weapons and to reload the weapons' ammunitions.

The 4 different weapons are subdivided in primary weapons and secondary weapons, all of the weapons scripts derives from the abstract class WeaponBase and the interface IWeaponUI, respectively used to set the weapon's parameters and to change them in the UI based on which one is equipped.
Thera are alse 4 different pickables ammo types based on the 4 different weapons, that use a trigger collider to be picked by the player and increase the current ammo the player has by a specific value.

- **PLAYER MOVEMENT**

To achieve the behavior written in the Game Design Document, the player movement system uses mathematical formulas (such as Mathf.Sin) to adjust the rotation, movement and oscillation of the player.
It is also added a function that permits the player to tilt the rotation to set it on a 90° rotation after the player stays still for a certain period of time.

- **METHODS**

- ■ RotationOnPrincipalAxes;
- ■ Movement;
- ■ Oscillation;
- ■ TiltOnRotation;

DATA SAVE SYSTEM

To achieve the behavior written in the Game Design Document, the static class SaveManager.cs is used to save the preferences of the screen, the audio and volume and the player prefs.

INTERACTABLES

To achieve the behavior written in the Game Design Document, the interactables system permits to open doors on certain conditions.
For the normal doors it was used a trigger collider to check the distance from the player in which the door opens and closes.
For the key doors it was added another condition, in which is checked if the player has the key to open a specific door.