# Script Documentation

All scripts are in the directory Assets 🡪 Scripts

Scripts are split into several folders.

## Camera

### CameraScript

* *How to use:* Attach this script to the camera object of a scene. Create game objects with the transform component under the camera as targets, e.g. camera moves from current position to target1, then moves from target1 to target2 etc. Give each target gameobject the tag ‘Camera Target’
* *Public Variables:*
  + speed (float): How fast the camera moves to one target to another
  + gameoverText (text): Text object to display gameover.
* *Summary:* The camera moves from one target to another starting from left to right at a given speed set. If the player touches the left-hand side of the camera, it results in a game over and restarts the level.

### levelOneCamera

* *How to use:* Attach this script to the camera object of a scene.
* *Public Variables*
  + *Player (Transform): The player that the camera follows*
* *Summary:* The camera follows the player, centred at the player.

### TutorialCamera

* *How to use:* Attach this script to the camera object of a scene.
* *Public Variables*
  + *Player (Transform): The player that the camera follows*
  + *left\_boundary (float):* Where the camera should follow the player horizontally after this x coordinate.
  + *bottom\_boundary (float): Where the camera should follow the player vertically after this y coordinate.*
* *Summary:* Similar to the levelOneCamera script, except that the camera only starts to follow the player horizontally when the player is in a position after the left boundary and follow the player vertically after the player is in a position more than the bottom boundary.

## Canvas

### Countdown Controller

* *How to use:* Attach to game controller.
* *Public Variables*
  + *countdownTime (Int):* Countdown time before game starts.
  + *countdownDisplay (Text):* Text that displays the countdown.
  + *rules (GameObject):* Rules shown on the canvas
  + *camera (GameObject): Camera*
  + *cameraScriptName (string):* Camera Script Name
  + *player (GameObject)*: Player
  + *playerscript (string):* Script of the player to move
* *Summary:* Sets a countdown at the beginning of the game level. During the countdown the rules is shown, and the players actions and camera script is disabled (in the case of a dynamic moving camera). After the countdown, the game starts, and the camera and player moves. The rules stop displaying.

## Character

### Enemy

#### enemy\_move

* *How to use:* Attach to enemy (i.e. red cube)
* *Public Variables*
  + leftPoint (Transform): How far left the enemy can go.
  + rightPoint (Transform) How far right the enemy can go.
  + speed (float): How fast the enemy moves.
* *Summary:* The player moves left and right up to the left and right point boundaries. The enemy pursues the player when nearby.

### Player

#### DeathScript

* *How to use:* Attach to player
* *Public Variables*
  + gameoverText (Text): Object containing text to display gameover
  + deathParticles (GameObject): Particle effect when player dies
* *Summary:* If the player touches anything with the specified tags in the scripts (i.e. traps, fireball, acid), a gameover is resulted and the level restarts. Before gameover the particle effects take into effect.

#### NewPlayerMovement

* *How to use:* Attach to player
* *Public Variables*
  + checkGround (Transform): The position under the character to identify when the player has touched the ground.
  + Platforms (LayerMask): The layer mask of the platforms
  + speed (float): How fast the player moves.
  + jumpSpeed (float): How fast the player jumps.
  + isGround (bool) – Current state of whether the player is on the ground.
  + isJump (bool) – Current state of whether the player is jumping.
* *Summary:* Handles the player movement. Handles when the player can double jump (i.e. when they have touched the ground previously). Also registers the players character state on start (see GameManager class).

#### playerFire

* *How to use:* Attach to the bullet/fireball prefab.
* *Public Variables*
  + speed (float): How fast the players bullet/fireball flies.
* *Summary:* When the fireball collides with the enemy, the enemy takes damage. This game object is also destroyed on collision.

#### PlayerWeapon

* *How to use:* Attach to player
* *Public Variables:*
  + fireball (GameObject): The prefab weapon i.e. fireball that is instantiated.
* *Summary:* instantiates fireballs when a keycode is pressed.

## Data

### CharacterData\_SO

* *Summary: extends from ScriptableObject. Takes a data file and exposes the public variables e.g. maxHealth, baseDefence, minDamage etc,* used elsewhere. An instance of this class is used as a public variable for the CharacterState class.

### CharacterState

* *How to use:* Attach to player
* *Public Variables*
  + templateData (CharacterData\_SO): instance of CharacterData\_SO
  + CharacterData (CharacterData\_SO): instance of CharacterData\_SO
* *Summary:* Contains a set of methods for handling the players stats (health, defence and position).
  + The public method ‘takeDamage’ reduces the players health, used when taking hits from the enemy.
  + Also encapsulates the public variables from CharacterData (i.e. getters and setters when needed).

### GlobalPrefs

* *How to use:* Static Class, not attached to game object
* *Public Variables*
* *Summary:* Contains methods for handling global data
  + public static void SaveCurrentLevel (int level): Takes a level and stores it in player prefs as the key ‘CurrentLevel’.
  + public static int LoadCurrentLevel: Loads the current level stored in player prefs for the key ‘CurrentLevel’. Returns the level.

### savePoint

* *How to use:* Attach to checkpoint
* *Public Variables*
  + playerData (CharacterState): instance of CharacterState
* *Summary:* Controls how a player saves their game i.e. by pressing E when near a checkpoint. See SaveManager class for saving the player data.

## Game Manager

### GameManager

* *How to use:* Attach to GameManager object of the scene
* *Public Variables:*
  + *playerStates (CharacterState)*
* *Summary:* Singleton class that holds the current player state, an object of the CharacterState class. Includes one public method ‘RegisterPlayer’, which sets the players state.

### SaveManager

* *How to use:* Attach to SaveManager object of the scene
* *Public Variables:*
* Summary: Singleton class with several public methods
  + SavePlayerData (void): Saves the current player’s character data (defined in CharacterState) using the playerState of the GameManager into the respective save file when registered.
  + LoadPlayerData(): Loads the data that has been saved from SavePlayerData.

## Items

### Keys

#### keys

* *How to use:* Attach to key object
* *Public Variables:*
* Summary: Increases the key quantity held by the character by 1, updating the KeyUI public static variable ‘CurrentKeyQuantity’.

#### KeyUI

* *How to use:* Attach to key object
* *Public Variables:* 
  + StartKeyQuantity (int): Number of keys at the start of the level
  + KeyQuantity (Text): UI component which reveals the key count.
  + CurrentKeyQuantity (int): Number of keys currently held by the player
* Summary: Updates the text of the canvas to display how many keys the player has.

#### treasureBox

* *How to use:* Attach to treasure box game object
* *Public Variables:* 
  + *doorKeys (GameObject): Prefab of the key*
* Summary:
  + When the player interacts with the treasure box (by pressing the key code E), a key is instantiated for the player to collect.

### DoorScript

* *How to use:* Attach to the exit door game object
* *Public Variables:* 
  + completeText (Text): UI component which displays the level is complete.
* Summary: Displays that the player has completed the level on colliding with the door. Moves to the next level or returns to the menu if all levels are complete.

### SpeedBoost

* *How to use:* Attach to the speed boost item
* *Public Variables:* 
  + rotationSpeed *(float)*: Purely aesthetic, the object rotates at a speed.
  + player *(GameObject): The* player *game object*
  + particles *(GameObject):* The prefab particle effects when receiving the speed boost
  + buffDuration (int): How long the boost should last for.
  + increaseSpeed (float): How much to increase the players speed by.
  + increaseJump (float): How much to increase the players jump by.
* Summary: Increases the characters speed and jump when the player has received a speed boost for a temporary duration. Instantiates particles to indicate the boost is in effect.

## Menu

### MainMenu

* *How to use:* Attach to the main menu object in the menu scene or the canvas main menu for the playable scenes.
* *Public Variables:*
  + mainMenu (GameObject): Parent game object containing all selectable menus, e.g. rules, options, etc.
  + rules (GameObject): Parent game object containing all the rules, canvas, UI objects etc.
  + options (GameObject): Parent game object containing all the options, canvas, ui objects etc.
* Summary: Contains various public methods used in the buttons of the menu.
  + playGame(): Called on clicking the play button of the menu scene to start the first level.
  + playTutorial(): Called on clicking the tutorial button of the menu scene to start the tutorial scene.
  + loadGame(): Called on clicking the load button. Loads the current level using the GlobalPrefs static method ‘LoadCurrentLevel’.
  + backToMainmenu(): Called on clicking the back button after viewing the options or rules, to return back to the main menu.
  + showRules(): Called on clicking the rules button to view the rules.
  + showOptions(): Called on clicking the options button to view the options.
  + resume(): Called to return back to the game if on a level scene.
  + quitGame(): Returns back to the menu scene.
  + restartLevel(): Restarts the current scene.

### MenuManager

* *How to use:* Attach to the Game Controller of a level scene
* *Public Variables:* 
  + gameMenu (GameObject)
* Summary: Allows the player to see the menu on hitting ‘esc’, and pauses the game temporarily.

## Options

### AudioSliders

* *How to use:* Attach to the volume sliders that control the audio (i.e. master, audio, effects).
* *Public Variables:* 
  + mixer (AudioMixer): The audio mixer used in the game.
  + volumeName: the name of the volume, e.g. master, audio, effects
  + volumeLabel (Text): Label of the slider
* Summary: Adjust the sound of a specified volume in the audio mixer based on the position of the slider.

### GameSettings

* Summary: Do not need to attach to a game object. This is a static class containing the current game settings, gamma etc.

### GammaSlider

* *How to use:* Attach to the gamma slider that controls the lighting
* *Public Variables:* 
  + *percentageLabel (Text): Label of the slider*
  + *globalLight (Light2D): The global light of the scene.*
* Summary:
  + Adjusts the intensity of the global light proportional to the position of the gamma slider. The default is 50% on the gamma slider (1.0.f light intensity). The gamma slider set to 100% is 1.5f light intensity and 0% set to the slider sets the light intensity to 0.5f. Intervals between are proportional.

### SettingsController

* *How to use:* Attach to the settings controller of the scene
* *Public Variables:* 
  + *masterAudioSlider (Slider): master audio slider*
  + *musicSlider (Slider): slider for the music volume*
  + *effectsSlider (Slider): slider for the effects volume*
  + *globalLight (Light2D): global light of the scene*
  + *countdownOnStart(Toggle)*
* Summary: Contains two public methods
  + saveSettings (void): When hitting the back button of the options menu, the settings are saved. (Saved in the ‘GameSettings’ static class)
  + loadSettings (void): Loads the positions of the game option sliders, checkboxes based on the ‘GameSettings’ public static class variables.

## Platform

### BreakingPlatform

* *How to use:* Attach to a one-time-use aka breaking platform.
* *Public Variables:*
* Summary: On the player colliding with the platform, the breaking animation plays and destroys the platform in 0.5 seconds.

### ElevatorPlatformVerticle

* *How to use:* Attach to an elevator platform
* *Public Variables:* 
  + topPoint (Transform): game object with transform component, the top boundary before the platform moves down again.
  + bottomPoint (Transform): game object with transform component, the bottom boundary before the platform moves up again.
* Summary:
  + Moves the platform up when it has reached the top point and move the platform down when it has reached the bottom point.

### HorizontalPlatform

* *How to use:* Attach to a platform that moves left or right when the player first stands on it.
* *Public Variables:* 
  + speed (float): the speed at which the platform moves.
  + end (Transform): the location at which the platform stops moving.
  + playerPlaceholder (GameObject): child gameobject of the platform, so the player moves with the platform.
  + moveLeft (bool): if checked, the platform moves left otherwise the platform moves right.
* Summary: The platform moves left or right when the player first stands on it and stops at an end point. It does not move back and forth between points.

### LevelOneMoveTransform

* *How to use:* Attach to a platform that moves left and right, back and forth.
* *Public Variables:* 
  + leftPoint (Transform): game object with transform component, the left boundary before the platform moves right again.
  + rightPoint (Transform): game object with transform component, the right boundary before the platform moves left again.
* Summary: Moves the platform left and right between two designated ends.

### SlipperyPlatform

* *How to use:* Attach to a slippery platform.
* *Public Variables:* 
  + leftPoint (Transform): game object with transform component, the left point which forces the player to slide towards if the player is facing to the left.
  + rightPoint (Transform): game object with transform component, the right point which forces the player to slide towards if the player is facing to the right.
* Summary: Performs lerp on the player to either the left or right point depending on where the player is facing. At Time.delta \* 1.

## Tools

### Singleton

* *How to use:* A class should inherit this singleton.
* Summary: Used if we only need one instantiation of a class. Used in GameManager and SaveManager.

## Traps

### AcidBlob

* *How to use:* Attach to acid blob
* *Public Variables:* 
  + *topPoint (Transform): position of where the acid blob should respawn*
  + *bottomPoint (Transform): position of where the acid blob should disappear.*
* Summary: An acid blob drops to the bottom point and then respawns at the top point again.

### Cannon

* *How to use:* Attach to cannon
* *Public Variables:* 
  + *fireball (GameObject): prefab object of fireball that respawns*
  + *respawnTime (int): Frequency at which fireball is shot from the cannon*
  + *killObjectTime (int): Frequency at which the fireball game object is destroyed after instantiation.*
* Summary: Instantiates/fires a fireball that hurts the player.

### SpikeTrap

* *How to use:* Attach to spinning blade/spike
* *Public Variables:* 
  + rotationSpeed (float): the speed at which the spike rotates.
* Summary: Purely aesthetic, allows the blade/spike to spin.