Hamina 1810 – How to

Contents

[What is this? 4](#_Toc103594614)

[Introduction on how to create duplicate IDs and break the game 4](#_Toc103594615)

[Duplicating Scriptable Objects, the Bad Way 4](#_Toc103594616)

[Duplicating Scriptable Objects Properly 5](#_Toc103594617)

[Useful Terminology 5](#_Toc103594618)

[Component 5](#_Toc103594619)

[Reference 6](#_Toc103594620)

[Parent 6](#_Toc103594621)

[Child 6](#_Toc103594622)

[This 6](#_Toc103594623)

[Space 6](#_Toc103594624)

[Local Space 6](#_Toc103594625)

[World Space 6](#_Toc103594626)

[Sprite Renderer 7](#_Toc103594627)

[Adding a game object to the scene 7](#_Toc103594628)

[Adding a component to a game object 8](#_Toc103594629)

[Adding an image to a game object 9](#_Toc103594630)

[Sorting Layers 9](#_Toc103594631)

[Order in Layer 9](#_Toc103594632)

[Settings 10](#_Toc103594633)

[Game Manager 10](#_Toc103594634)

[Dialog Manager 10](#_Toc103594635)

[Character Selection 11](#_Toc103594636)

[Tarot Card 11](#_Toc103594637)

[Character Build Menu 12](#_Toc103594638)

[Player 12](#_Toc103594639)

[Lighting Manager 13](#_Toc103594640)

[Work 13](#_Toc103594641)

[Colliders and triggers 15](#_Toc103594642)

[Box Collider 2D 15](#_Toc103594643)

[Circle Collider 2D 16](#_Toc103594644)

[Polygon Collider 2D 17](#_Toc103594645)

[Custom Components 18](#_Toc103594646)

[Object ID (Important!) 18](#_Toc103594647)

[Object Base Mono 19](#_Toc103594648)

[Dynamic Sort 20](#_Toc103594649)

[Dynamic Sort Static 20](#_Toc103594650)

[Dynamic Sort Static Advanced 23](#_Toc103594651)

[Dynamic Sort Moving 25](#_Toc103594652)

[Dialog Triggers 25](#_Toc103594653)

[Dialog Box 25](#_Toc103594654)

[Animation Controller 26](#_Toc103594655)

[Interaction Mark Offset 26](#_Toc103594656)

[NPC 26](#_Toc103594657)

[Location 27](#_Toc103594658)

[Proximity Highlight 27](#_Toc103594659)

[Rotate 28](#_Toc103594660)

[Tooltip 28](#_Toc103594661)

[Street Light 28](#_Toc103594662)

[Light Flicker 28](#_Toc103594663)

[Custom UI Components 29](#_Toc103594664)

[Flexible Grid Layout Group 29](#_Toc103594665)

[Game Action Click Trigger 30](#_Toc103594666)

[Open Link On Click 30](#_Toc103594667)

[Scriptable Objects 30](#_Toc103594668)

[Application Background 31](#_Toc103594669)

[CV 32](#_Toc103594670)

[Dialog Data 33](#_Toc103594671)

[Dialog 33](#_Toc103594672)

[Education 34](#_Toc103594673)

[Game Action 35](#_Toc103594674)

[Game Event 45](#_Toc103594675)

[Highlight 46](#_Toc103594676)

[Item 46](#_Toc103594677)

[Job Application 47](#_Toc103594678)

[Keywords 48](#_Toc103594679)

[Application Reference 48](#_Toc103594680)

[Background Story 48](#_Toc103594681)

[Employer Description 48](#_Toc103594682)

[Filler Phrase 48](#_Toc103594683)

[Greeting 48](#_Toc103594684)

[Job Description 48](#_Toc103594685)

[Motto 48](#_Toc103594686)

[Player Description 48](#_Toc103594687)

[Player Goal 49](#_Toc103594688)

[Valediction 49](#_Toc103594689)

[Loot Table 49](#_Toc103594690)

[Overlay Properties 49](#_Toc103594691)

[Tiled Overlay Properties 49](#_Toc103594692)

[Popup Container 50](#_Toc103594693)

[Professional Field 51](#_Toc103594694)

[Profession 51](#_Toc103594695)

[Quest Category 53](#_Toc103594696)

[Quest 53](#_Toc103594697)

[Random Event 54](#_Toc103594698)

[Random Event Trigger Table 54](#_Toc103594699)

[Random Event Trigger Table Item 54](#_Toc103594700)

[Random Work Event Table 55](#_Toc103594701)

[Random Work Event Table Item 55](#_Toc103594702)

[Workday 55](#_Toc103594703)

[Work Start Stage 55](#_Toc103594704)

[Work End Stage 56](#_Toc103594705)

[Work Start Option 56](#_Toc103594706)

[Work Challenge Option 56](#_Toc103594707)

[Work Lunch Option 56](#_Toc103594708)

[Sprite Animation 57](#_Toc103594709)

[Trait 57](#_Toc103594710)

[Trigger 57](#_Toc103594711)

[Other Useful Classes 58](#_Toc103594712)

[Dynamic Text 58](#_Toc103594713)

[Interaction Mark 59](#_Toc103594714)

[Prerequisites 60](#_Toc103594715)

# What is this?

This document is an instruction manual on how to modify or add content in the Hamina 1810 Unity project. The project was made in a way, that allows people with no programming experience to add new and modify existing content within the project.

Note that while the project gives you quite a few possibilities when adding or editing content, **changing the UI hierarchy will most likely break the game**, and will require changing the code. Changing UI images is fine as long as the UI hierarchy (UI object names and positions within the hierarchy) is not changed.

# Introduction on how to create duplicate IDs and break the game

One thing to keep in mind are the Object IDs. Many [Custom Components](#_Custom_Components) and most [Scriptable Objects](#_Scriptable_Objects) are automatically assigned an ID when the game is run. The ID is an important way of identifying an object and it must be unique to the object type or things may start breaking.

## Duplicating Scriptable Objects, the Bad Way

While duplicating objects is generally fine, there are situations where doing so will sometimes break things. A scriptable object is automatically assigned an ID when the game is run if the scriptable object is located in the folder specified later in this document. This ID is often used to find the object and retrieve it through the Object Manager.

Now, if a scriptable object already has an ID and it is then duplicated, the duplicate object will have the same ID as the original. This is bad! In some cases, Dialog Triggers might start accessing the wrong Dialog objects and then you’ll have an NPC in the game that doesn’t look like themselves and has stolen all their lines from another NPC. As far as I know, this problem only occurs with the Dialog and DialogData objects, since they store data of who’s referencing them in order for them to be able to start a dialog without needing an additional reference to a DialogTrigger. But to be on the safe side, I’d read on and follow the instruction below on how to avoid most problems caused by object duplication.

For GameObjects (the objects in the scene, listed in the hierarchy) this is a bit different since the automated system saves the creation time of the object (if the object/component is set to be [persistent](#_Object_ID_(Important!))). When an ID collision is detected, the system compares the creation times, and the older object keeps their ID. The possible problems start when one of the objects is a duplicate with identical ID and creation time. Now the Object Manager doesn’t know which one is older and therefore lets the object that was loaded first keep their ID. If the object that was loaded first happens to be the duplicate, all ID references that were pointing to the original object are now pointing to the duplicate.

## Duplicating Scriptable Objects Properly

If you need template scriptable objects, you should create a template folder outside of the Resources folder and save all the templates (scriptable objects WITHOUT an ID) you want to duplicate later, in there. This way the automatic systems will not load the scriptable objects, nor will they assign an ID or even know your scriptable object exists.

After duplicating from a template, you can move the file to the folder specified later in this document (each scriptable object type has their own folder). Afterwards, when the game is run, the object will be assigned a fresh new unique ID.

# Useful Terminology

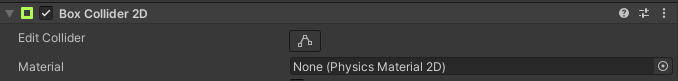
## Component

A component in Unity is a script that is attached to a GameObject. Each component has its own job, which can be anything from storing data to complex algorithms.

Graphical user interface, application

Description automatically generatedComponents attached to a GameObject are listed in the Inspector window when the GameObject is selected. The GameObject, in the image below, has two components attached to it: Transform and Sprite Renderer.

## Reference

Reference is a term used when a component refers to another component or object. E.g., this Box Collider 2D component (below) can take a Physics Material 2D reference but isn’t currently referring to one.

### Parent

A parent object is an object that has at least one child object attached to it. In the hierarchy, this is visualized with the small arrow next to its name.

In the image below, ‘WindowContainer\_Panel’ is the parent of the ‘Window\_Panel’ object.

***Note: When moving or resizing a parent object, all of its children will be moved and/or resized as well.***

### Child

The term “child” refers to a game object that is attached to a parent object. In the image above, ‘Window\_Panel’ is a child of the ‘WindowContainer\_Panel’ object.

### This

In this document, the terms “this” is sometimes used to refer to an instance of the component itself, or a game object that the component is attached to. E.g., “this component” or “this game object”.

## Space

### Local Space

Local space is the space relative to an object. E.g., child objects inhabit the local space of their parent.

### World Space

World space is the scene’s space. All root objects (objects without a parent object) exist in world space.

## Sprite Renderer

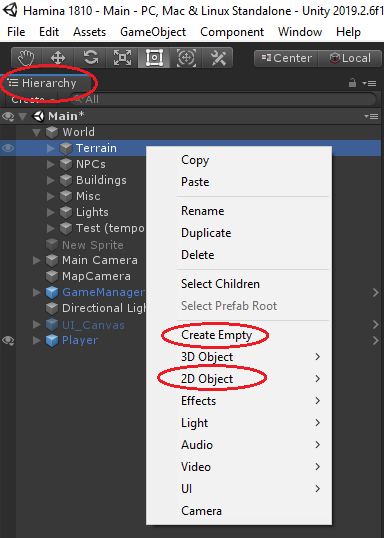
A component attached to a GameObject that is used to render a sprite in the scene in world space.

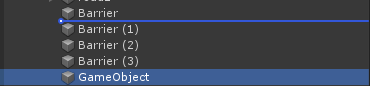
# Adding a game object to the scene

There are multiple ways of going about this but here are a couple of examples.

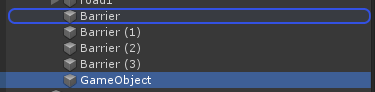
Locate the object - to which you want to attach the new object - in the hierarchy and left-click first to select it, then right-click it and make you selection based on the following:

If the object…

* …will not have an image attached to it, select Create Empty.
* …will have an image attached to it, select 2D Object > Sprite.

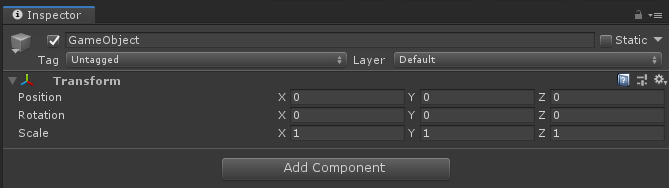
After creating the new object, you should make sure it’s located in the correct place in the hierarchy. Objects can be moved around in the hierarchy by dragging and dropping. Dropping an object when a line is visible between two other objects moves the dropped object between the two objects.

In this case (above), ‘GameObject’ would be moved below ‘Barrier’ but would NOT be attached to it.

Dropping an object when another object’s name is highlighted with a line around its name makes the dropped object a child of the highlighted object. In this case (below), ‘GameObject’ would become a child of the ‘Barrier’ object.

Now, remember to rename the object. Renaming the object can be done by selecting it in the hierarchy and pressing F2 or right clicking the object (in the hierarchy) and selecting Rename.

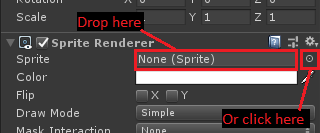
# Adding a component to a game object

Select the target object from the hierarchy by left clicking it and locate the Inspector tab/window. The Inspector window is often located on the right side of the main UI.

Make sure the object doesn’t already have that type of component attached to it and click Add Component. Type the component name in the search field and left-click it in the menu to add it.

# Adding an image to a game object

Add a Sprite Renderer component to the object if it doesn’t already have one.

Select a Sprite for the Sprite Renderer component by either dragging and dropping it from the Project files or clicking on the small circle and selecting it from the popup menu.

## Sorting Layers

Next, select the right Sorting Layer for the object. The layers are drawn starting from the first layer in the menu, then the second layer is drawn on top of that and so on. This means that the first layer is under of everything, and the last layer is above everything when all layers have been drawn.

* **Background:** Furthest from the camera, behind everything.
* **Default:** Not in active use. The default layer where new objects live. **Note:** If you can’t see an object, it’s most likely behind the terrain.
* **Terrain:** Used for terrain and other objects (like roads) that are always drawn behind the player and all other objects in the world.
* **Below Dynamic Sort:** A layer for the cases when the object is not part of the terrain, but should be draw between the terrain and the main layer.
* **Dynamic Sort:** This is the main layer of the game. Most objects are located on this layer and they are dynamically sorted based on their locations relative to each other. ([Read more about Dynamic Sort](#_Dynamic_Sort))
* **Above Dynamic Sort:** Like the name hints, objects on this layer will be drawn on top of the objects on the main layer (Dynamic Sort).
* **InteractionMarks:** This layer is for the automatically generated interaction marks and should not be used for anything else.
* **UI:** Not in active use.
* **Menu:** Not in active use.

## Order in Layer

This is where the drawing order of the layer is determined. Drawing starts from the smallest number. So -700 is drawn before 200 and therefore is behind. If an object uses the [Dynamic Sort](#_Dynamic_Sort) system, the Order in Layer is assigned automatically and can therefore be ignored.

# Settings

This section goes through all the game settings and where to find them.

## Game Manager

The GameManager game object can be found in the root of the hierarchy. It contains a component that is also called GameManager, this component includes the following settings:

* **START\_IN\_MAIN\_MENU:** If checked, the game starts normally in the main menu. If not checked, jumps straight into the game skipping the character creation. Useful for quick testing and/or debugging.
* **DEBUG:** Turns on debugging mode. Turning debugging mode off disables some log messages and Object Manager log file.
* **DEVELOPER\_MODE:** The developer mode allows cheating: give money, no-clip, etc.
* **Current Game State:** asd
* **CONTEXT\_MENU\_AUTO\_CLOSE:** If checked, the context menu (opened by right-clicking on some UI elements) closes automatically after the cursor has been outside of the context menu for an amount of seconds set in **CONTEXT\_MENU\_CLOSE\_TIMER**.
* **CONTEXT\_MENU\_CLOSE\_TIMER:** The time (in seconds) it takes the context menu to automatically close when **CONTEXT\_MENU\_AUTO\_CLOSE** is enabled and cursor is not on top of the context menu.
* **Random Event Check Time:** The interval (in seconds) between random event checks. E.g., If set to 60, a random event can trigger once every minute.
* **Random Event Base Chance:** The chance to trigger a random event. The actual event is randomly chosen AFTER this triggers first. E.g., If set to 0.02, the chance to trigger a random event is 2%.
* **In Game Time:** The in-game time the game starts at.
* **Total Time:** The total amount of time the game has been running. **Note:** Not meant to be edited. *In the future these timers should be changed to a proper timekeeping system that automatically updates times based on game and player states.*
* **Play Time:** The amount or time the game has been in PLAYING state. **Note:** Not meant to be edited.
* **Dialog Time:** The amount of time the player has spent in dialogs. **Note:** Not meant to be edited.
* **Menu Time:** The amount of time the player has spent in menus. **Note:** Not meant to be edited.
* **Interaction Marks Use Icons:** If checked, the floating interaction marks use interaction specific icons. If not checked, all icons will be different color exclamation marks.
* **Rent Properties:** Defines the global rent settings:
  + **Days Between Rents:** The number of days before rent is collected after renting an apartment.
  + **Time Of Rent Payment:** The time of day of the rent collection. [Game Time](#_Game_Time), ignores days.
  + **Rent Paid Popup:** A Popup Container, displayed when the rent is paid successfully.
  + **Rent Not Paid Popup:** A Popup Container, displayed when the rent is not paid.
  + **Debt Collection Popup:** A Popup Container, displayed when the rent has been unpaid for Days Between Rents times 2. If Days Between Rents is set to 3, then 3 \* 2 = 6.

## Dialog Manager

This component manages dialogs and the dialog UI. By default the component can be found int the GameManager object.

Dialog Manager contains the following properties:

* **Job Interview Dialog Header:** The sprite that will be used as the dialog window header in a job interview.
* **Job Interview Dialog Box Background:** The sprite that will be used as the dialog window background in a job interview.
* **Disable Reactions:** If checked, dialog reactions will be disabled.
* **Reaction Image Fade Time:** The default time it takes for the reaction image to fade out.

## Character Selection

The character selection component can be found from the hierarchy in an object called ‘CharacterSelectionMenu\_Panel’. It controls the tarot card positions and scales in the character selection screen. The tarot cards will return to their original positions and sizes when not selected. The following settings can be changed from the character selection component:

* **Hover Speed:** The speed at which the tarot cards are rescaled when hovered over.
* **Select Speed:** The speed at which the tarot cards are rescaled when selected.
* **Reset Speed:** The speed at which the tarot cards are rescaled when deselected.
* **Hover Scale:** The target scale of the tarot card when hovered over.
* **Selected Scale:** The target scale of the tarot card when selected.
* **Selected Position:** The target position of the tarot card when selected.

## Tarot Card

Tarot cards are used in character creation. The tarot card UI elements can be found in the Assets/Resources/Prefabs/UI/Canvases folder by editing the ‘MainManu\_Canvas’ prefab. The TarotCards are children of a GameObject called ‘TarotContainer\_Panel’. The number of tarot cards can be changed by removing or adding/cloning the UI elements that contain a TarotCard component. Tarot card contains the following properties:

* **Is Female:** Determines if a character is female. Only used in random name generation.
* **Character Dialog Image:** An image (Sprite) of the character that will be used in the dialog view.
* **Character CV Portrait:** An image used in the CV as a portrait.
* **Idle Animations:** A list of idle animation objects ([SpriteAnimationSO](#_Sprite_Animation_1)) for the character.
* **Moving Animations:** A list of moving animation objects ([SpriteAnimationSO](#_Sprite_Animation_1)) for the character.
* **Character Description:** A description of the character. Visible on the character selection screen.
* **Character Point Modifier:** A modifier for character points. E.g. If set to 3 and [base points](#CharacterBuildMenu_PointsToSpend) are set to 5, the amount of points the player gets to spend on the character is 8. If base is 5 and modifier is -3 the player gets to spend 2 points.
* **Social Class:** The social class of the player. Can affect traits that the character can acquire.
* **Starting Attributes:** The default attributes of the character.
* **Starting Resources:** The default resources of the character.
* **Starting Traits:** The traits ([TraitSO](#_Trait)) the character starts with.
* **Starting Money:** The amount of money the character starts with.
* **Starting Items:** The items ([ItemSO](#_Item)) the character starts with.

## Character Build Menu

The character build menu component can be found from the hierarchy in the UI element called ‘CharacterBuildMenu\_Panel’. It contains the following properties:

* **Card Speed:** The speed at which the tarot card moves from [Character Selection](#_Character_Selection) [Selected Position](#CharacterSelection_SelectedPosition) to character build menu [Target Tarot Position](#CharacterBuildMenu_TargetTarotPosition).
* **Target Tarot Position:** The position to where the selected tarot card will be moved from [Character Selection](#_Character_Selection) [Selected Position](#CharacterSelection_SelectedPosition).
* **Target Tarot Scale:** The scale the tarot card will be resized to.
* **Points To Spend:** The base amount of character points the player gets to spend on the selected character.
* **Positive Trait Image:** The icon that will be visible next to all positive traits.
* **Negative Trait Image:** The icon that will be visible next to all negative traits.

## Player

Player settings can be found in the ‘Player’ object.

**Note:** Some of the properties in the Player component are meant to be used while debugging or testing and will only affect the game if changed while the game is running. These properties will be marked with a **-DEBUG** suffix.

The player component includes the following properties:

* **Character State-DEBUG:** Current character state.
* **Player Name-DEBUG:** Player’s name.
* **Walk Speed:** The character’s walking speed.
* **Run Speed:** The character’s running speed.
* **Starvation Damage Per Hour While Sleeping:** Health points per hour. The amount of damage the player takes while sleeping and starving at the same time.
* **Starvation Damage Per Second:** Health points per second. The amount of damage the player takes when walking/idling and starving.
* **Player Dialog Image-DEBUG:** The player’s image shown in the dialog view.
* **Player Tarot Image-DEBUG:** The player’s image shown in the inventory window.
* **Tarot Card-DEBUG:** A reference to a tarot card.
* **Social Class-DEBUG:** Player’s social class.
* **Attributes-DEBUG:** Player’s attributes.
* **Resources-DEBUG:** Player’s resources.
* **Inventory Space:** The number of inventory slots the player has.
* **Active Traits-DEBUG:** Player’s active traits.
* **Home:** Player’s home location.
* **Consecutive Debt Collection-DEBUG:** The number of times the [debt collection](#DebtCollection) action has been triggered consecutively.
* **Last Debt Collection Time-DEBUG:** Last in-game time the [debt collection](#DebtCollection) action was triggered.
* **Death Message Hunger:** The message shown in the pop-up when player dies because of starvation.
* **Death Message Health:** The message shown in the pop-up when player dies to losing all their health.
* **Death Message Sanity:** The message shown in the pop-up when player dies to low sanity.
* **CV SO:** Player’s starting CV.

## Lighting Manager

By default, the lighting manager can be found from the hierarchy in the ‘Sun’ object. The Lighting Manager must be accompanied by a Light 2D component (Light Type: Global) for it to work properly. It automatically controls the day/night cycle and the streetlights based on the following properties:

* **Sunrise Time:** The in-game time at which the sun rises.
* **Sunset Time:** The in-game time at which the sun sets.
* **Sunrise Light Intensity:** The intensity of global illumination at the time of sunrise.
* **Day Light Intensity:** The intensity of global illumination at the time of day (exactly in the middle of sunrise and sunset).
* **Sunset Light Intensity:** The intensity of global illumination at the time of sunset.
* **Night Light Intensity:** The intensity of global illumination at the time of night (exactly in the middle or sunset and sunrise).
* **Use Daylight Colors:** If checked, the 4 colors below will be used to change the color of light. If not checked, the 4 colors below will be ignored.
* **Sunrise Color:** The color of global illumination at the time of sunrise. **Note: The colors automatically blend from one to the next based on the time of day.**
* **Day Color:** The color of global illumination at the time of day.
* **Sun Set Color:** The color of global illumination at the time of sunset.
* **Night Color:** The color of global illumination at the time of night.
* **Streetlight Properties:** Controls when [streetlights](#_Street_Light) are turned on and off.
  + **Time Type:** The way the time streetlights are turned on and off is calculated.
    - **MINUTES:** Absolute, e.g., 60 minutes would turn the lights on 60 minutes before sunset and turn them off 60 minutes after sunrise.
    - **PERCENTAGE:** Relative, e.g., 0.05 percent would the lights on when there's 5 percent of the day left (before sunset) and turn the light off when 5 percent of the day has passed (after sunrise).
  + **Minutes:** In-game minutes. Used when **Time Type** is set to **MINUTES**.
  + **Percent:** Percentage of time. Used when **Time Type** is set to **PERCENTAGE**.

## Work

The Work component can be found in the canvas prefab ‘Work\_Canvas’, that in turn is in the ‘Assets/Resources/Prefabs/UI/Canvases’ folder. The Work component handles the player’s workdays. A workday is split in to 4 sections: Start, Lunch, Challenge (optional/random) and End stages. Each of these stages have their own settings and functions:

* **Start:** In this stage the game informs the player if they’re late for work. The player is presented with Work Start Stage Options that determine how much resources the player uses.
* **Lunch:** In this stage the player is presented with lunch options that includes all consumable items in their inventory and any Lunch Stage Options added to the Workday.
* **Challenge:** An optional stage that is randomly picked from a [Random Work Event Table](#_Random_Work_Event).
* **End:** This stage informs the player how much they were paid for the workday. If the player was late, gives them the option of compensating for being late.

The component contains the following properties:

* **Negative Start:** The color of the first section of the resource bar if the resource change is negative.
* **Negative End:** The color of the second section of the resource bar if the resource change is negative.
* **Positive Start:** The color of the first section of the resource bar if the resource change is positive.
* **Positive End:** The color of the second section of the resource bar if the resource change is positive.
* **Lower Class Frame:** The image frame for lower class.
* **Middle Class Frame:** The image frame for middle class.
* **Upper Class Frame:** The image frame for upper class.
* **Late Text:** The [Dynamic Text](#_Dynamic_Text) that is displayed if the player arrives to work late.
* **Points Lost Per Hour Late:** The profession points the player loses for each hour they are late for work.
* **Lunch Stage Text:** The [Dynamic Text](#_Dynamic_Text) that is displayed in the lunch stage.
* **Late End Stage Text:** The [Dynamic Text](#_Dynamic_Text) that is displayed in the end stage if player arrived late to work.
* **Compensate For Tardiness Button Text:** The [Dynamic Text](#_Dynamic_Text) that is displayed on the compensate button.
* **Reaction Limits:** Optional. The reactions can be used to visually show the player how well they’re doing their job. When the player’s profession points are within a given range, an image will be displayed in a speech bubble above the employer.
  + **Min:** The minimum value of the range (inclusive).
  + **Max:** The maximum value of the range (inclusive).
  + **Sprite:** The image that will be displayed.
* **Default Point Actions:** A set of [Game Actions](#_Game_Action_1) that are triggered based on the player’s profession points if the [profession](#_Profession) doesn’t have any point actions set.
* **Low Resources Popup:** A [popup](#_Popup_Container) object that will be displayed if the player doesn’t have enough resources to work.
* **Already Worked Today Popup:** A [popup](#_Popup_Container) object that will be displayed if the player has already worked today.
* **Too Late To Work Popup:** A [popup](#_Popup_Container) object that will be displayed if the player came to work too late and can’t start work.
* **Start Work Early Popup:** A [popup](#_Popup_Container) object that will be displayed if the player came to work early.
* **No Work Today Popup:** A [popup](#_Popup_Container) object that will be displayed if the player tries to work on a day when the workplace isn’t open.

# Colliders and triggers

Colliders are always attached to a game object and can be added [like any other component](#_Adding_a_component). Colliders are used (in this case) to restrict player movement. They can also be used as triggers by checking the Is Trigger property in the collider’s properties. Be aware that doing this disables collisions for the collider in question.

Be sure to always use a 2D version of any collider you choose.

## Box Collider 2D

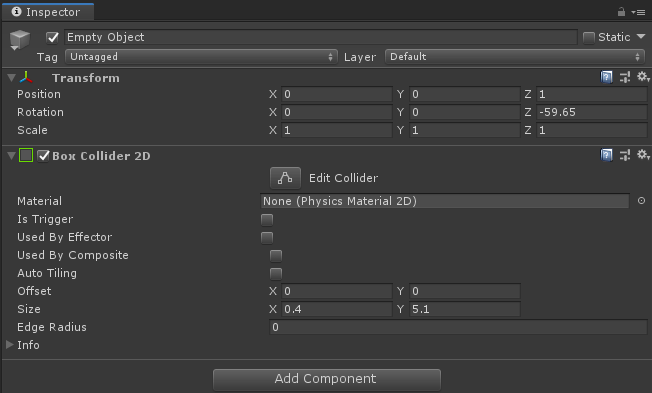
A rectangle shape collider which is good enough for most cases.

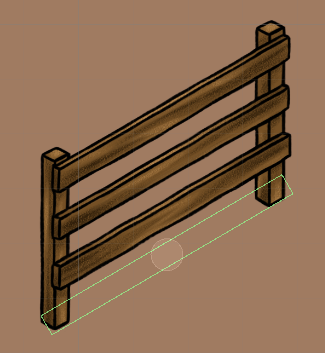
The position of the collider can be changed with the Offset X and Y values or by clicking on the Edit Collider button and dragging from the squares that appear on the collider.

Tip: The Box Collider can be rotated if it’s, for example, attached to an empty object and then the empty object is rotated on the Z-axis.

**Example**:

Here we have a ‘Fence’ object, and it has an empty game object as its child.

The empty game object is the one that contains the collider.

By changing the size of the collider and the position and rotation of the empty game object we can achieve something like this:

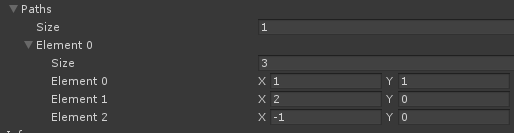
## Circle Collider 2D

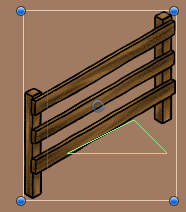
Circle Collider is a lot like the Box Collider but has Radius instead of Size. Circle Collider isn’t affected by its object’s rotation or scale.

## Polygon Collider 2D

Polygon Colliders are the most complex of the three. They can be used to make shapes that are not possible with other colliders.

When adding a Polygon Collider to a game object with a sprite attached to it, Unity automatically creates a shape for the collider based on the sprite’s shape. This is usually not what is required, so to erase the collider and start from scratch you can open the Points section in the collider component. Under that you’ll find an array called ‘Paths’. Change the Size of the Paths array to 0, press enter and change it up to 1 again. Now the collider is gone, and we can start working on a new one.

You should see another array under Paths called ‘Element 0’. Change the size of this array to 3 and insert some random values (not too big ones) to the X and/or Y fields and you should see something like this:

And in the Scene view:

Now you’re ready to edit the collider. Click on the Edit Collider button and hover the cursor over the collider. You’ll notice that the collider’s lines highlight (they get thicker) based on where the cursor is.

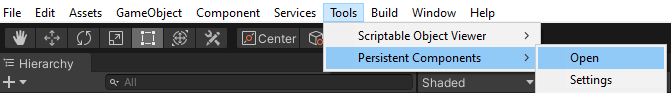
When one line is highlighted, a new point can be added by clicking and dragging.

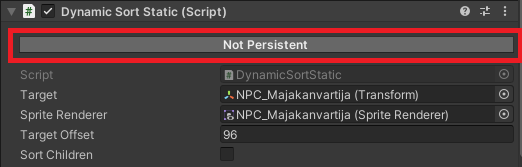


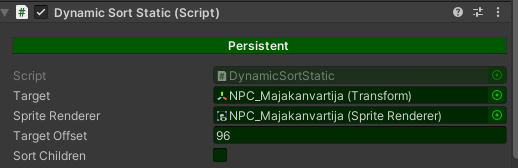
When moving the cursor next to a corner, two lines will get highlighted. This is when a point can be moved to another position by clicking and dragging.

# Custom Components

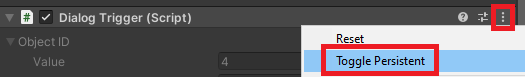
## Object ID (Important!)

Most custom components coded specifically for Hamina 1810 require an Object ID. The ID is given to the object automatically when the game is run **but** Unity does not save changes made in play mode by default! Persistent Components by BrokenVector was added to the project for this purpose.

To save an ID, the component or GameObject must be set to persistent before running the game. This can be done by clicking the ‘Not Persistent’ button in the component (if visible):

The component will be tinted green when persistent.

If the persistency button is not visible use one of the two the methods below:

* Open the Persistent Components window from Tools > Persistent Components > Open. Then drag & drop the GameObjects or components to the window from the hierarchy and/or the inspector.
* Or click the 3 dots in the top right corner of a component and select ‘Toggle Persistent’ 

Then all you need to do is run the game and make sure the ID was saved after stopping the game. Afterwards it’s best to set objects back to being non-persistent.

## Object Base Mono

This component’s only function is to work as a reference for other components or systems, e.g., [Game Actions](#_Game_Action_1).

There are different types of ObjectBaseMono components. The following components inherit the ObjectBaseMono class:

* [Character](#_Character)
* [Dialog Trigger](#_Dialog_Triggers)
* [Location](#_Location)
* [NPC](#_NPC)
* [Player](#_Player)
* [Proximity Highlight](#_Proximity_Highlight)

## Dynamic Sort

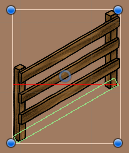
Dynamic Sort is the sprite sorting system designed for this game. It’s designed to sort the game objects in the scene so that the world feels 3-dimensional. This illusion is achieved with the following three components. Dynamic Sort is added to an object like any other component (see [Adding components](#_Adding_a_component)).

### Dynamic Sort Static

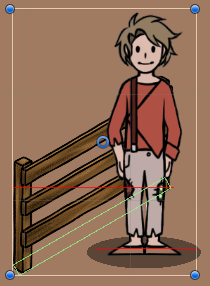
This is the simplest of the three and also the lightest. It’s only updated once during launch unless forced to update by a [Dynamic Sort Static Advanced](#_Dynamic_Sort_Static) component or if moved with Object Translations (see [Game Actions](#_Game_Action)). This component is designed to be used with static objects or object that are only moved with [Object Translations](#ObjectTranslation).

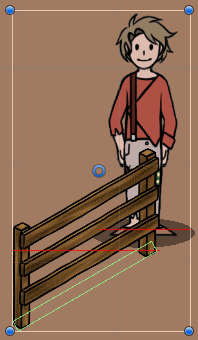
The sorting position is calculated from the object’s Y-coordinate and Target Offset.

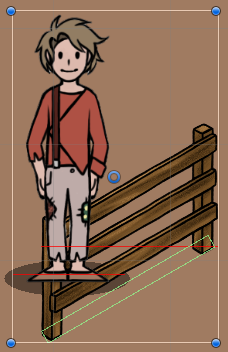
This component works best with objects that have only one point on the Y-axis where the change - from being behind to being in front - happens. This tree is a good example: When the player crosses the line so that they’re above it, they will be considered to be behind the tree.

This fence, on the other hand, is a tricky one.

If used in this manner, this is what happens when the player moves near the fence:

When the player is “in front” of the fence, the objects are drawn as expected since the player’s red line is below the fence’s red line.

When the player is behind the higher post, the objects are still drawn as expected.

When the player moves behind the lower post, things start to break.

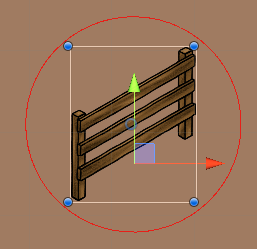
The player’s red line is below the fence’s line and so the player is considered to be in front of the fence. This could be solved with a larger collider, adding other objects with colliders behind the fence, blocking the player from moving to the problematic position or by using the [Dynamic Sort Static Advanced](#_Dynamic_Sort_Static) component.

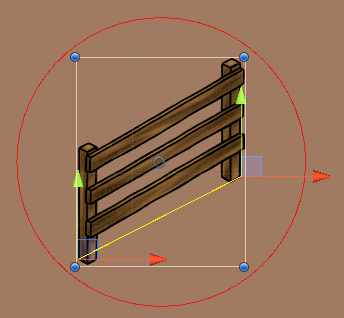
The Dynamic Sort Static component contains the following visible properties:

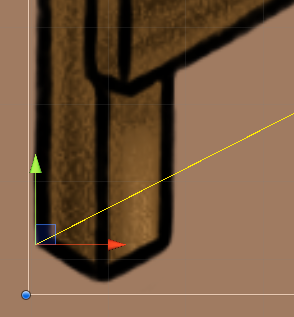
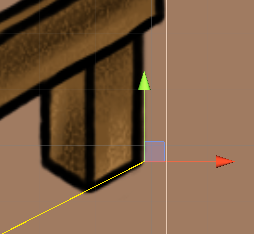
* **Target:** This is the transform that is used to calculate the object’s position in the sorting order. If not changed it will be automatically assigned to the object’s own transform. This can be changed by just dragging and dropping the wanted object from the hierarchy to the Target slot.
* **Sprite Renderer:** The target Sprite Renderer who’s sorting order position is being manipulated. Can also be changed by dragging an object from the hierarchy.
* **Target Offset:** The most important part of the component. Determines the object’s position in the sorting layer. When the object is selected the offset is visualized with a red line in the editor. Simply put whichever object’s line is higher, will be drawn first and therefore will appear to be behind the others.
* **Sort Children:** Sorts the children of the object in the order they’re in the Hierarchy.

### Dynamic Sort Static Advanced

The most resource hungry of the three and therefore should not be used unless absolutely needed.

This component checks the player’s distance to itself and sorts itself and any overlapping objects with a [Dynamic Sort Static](#_Dynamic_Sort_Static_1) component when the player comes within the Sorting Distance (visualized by the red circle in the editor).

The two Points (Left and Right) are used to determine when the player is behind the object (visualized by the two red-green handles and a yellow line between them).

The left point should be positioned in the leftmost corner/point of the sprite where the player would go behind the object. The same goes for the right point but on the right side.

The points can be moved by typing in the coordinates through the Inspector or by dragging the handles from the arrows or the blue rectangle.

Dynamic Sort Static Advanced properties:

* **Target:** see [Dynamic Sort Static](#DynamicSortTarget)
* **Sprite Renderer:** see [Dynamic Sort Static](#DynamicSortSpriteRenderer)
* **Target Offset:** Can be used to offset the sorting but most often left to 0.
* **Sort Children:** Sorts the children of the object in the order they’re in the Hierarchy.
* **Sorting Distance:** The distance to the player at which the component activates and starts sorting itself and any overlapping objects with a [Dynamic Sort Static](#_Dynamic_Sort_Static_1) component attached to them.
* **Points:** Used to define the points at which the player goes behind (or in front) of the object.

Dynamic Sort Static Advanced can also sort its child objects. This is done in the order of the hierarchy – each child’s sorting order is incremented by 1 from the last one. **Note:** Using this could have unwanted side effects if the margin of error is small.

### Dynamic Sort Moving

The Dynamic Sort Moving component is almost completely identical to [Dynamic Sort Static](#_Dynamic_Sort_Static_1) except for one thing; its sorting position is actively recalculated.

This component was originally designed to only be used by the player object.

## Dialog Triggers

Dialog Triggers are used to trigger either dialogs or a [Game Action](#_Game_Action_1).

* **Triggered With:** Determines the way the dialog is triggered.
  + **NONE**: Can only be triggered through a [Game Action](#_Game_Action).
  + **COLLISION:** Triggers when player collides with this object’s collider. Requires a collider.
  + **COLLISION\_AND\_TRIGGER:** Triggers when player collides with this object’s collider and the Action button is pressed (default: spacebar).
  + **BUTTON:** Triggers when player is in the trigger area and the Action button is pressed. Requires a trigger (see [Colliders and triggers](#_Colliders_and_triggers)).
* **DialogDataSO:** A Dialog Data object (see [Dialog Data](#_Dialog_Data)).
* **GameActionSO:** Can be used instead of a DialogDataSO if interacting with the object has to trigger an action (GameActions can start a dialog).
* **Create Interaction Mark:** Determines if a floating mark (to guide the player) should be automatically created above the object containing the Dialog Trigger.

## Dialog Box

The Dialog Box component is, by default, attached to ‘DialogBoxContainer\_Panel’ object in the ‘Dialog\_Canvas’ prefab.

This component stores properties regarding the dialog window:

* **Animation Type:** Determines how the dialog box behaves when starting or stopping dialog.
  + **MOVEMENT:** Dialog box is moved up from “under” the screen when opened, and back down when closed.
  + **FADE:** Dialog box fades in and out.
* **Animation Speed:** The speed of the movement or fade.

## Animation Controller

The animation controller can be used with characters and with non-character objects. It automatically animates the object when it moves or idles, based on the referenced [SpriteAnimationSOs](#_Sprite_Animation). The Animation Controller is added to object like any other component, see [Adding a component](#_Adding_a_component).

* **Idle Animations:** The animation that is played when the animated object isn’t moving.
* **Moving Animations:** The animations that are played when the object is moving. Contains 8 animations slots for 8 directions of movement. Some of the animation slots can be left empty, e.g., if the movement is 4-directional only the main directions (up, down, right & left) can be used. The indices are bound to certain directions:
  + **Element 0:** North/Up
  + **Element 1:** Northeast
  + **Element 2:** East/Right
  + **Element 3:** Southeast
  + **Element 4:** South/Down
  + **Element 5:** Southwest
  + **Element 6:** West/Left
  + **Element 7:** Northwest
* **Time Before Idle Animation:** The amount of time (in seconds) after stopping when the idle animation starts playing.

## Interaction Mark Offset

A picture containing text, clipart

Description automatically generatedThis component is used to change the default position of an Interaction Mark. The [interaction mark](#_Interaction_Mark) (image below) is an automatically generated element to help guide the player.

## NPC

The NPC component is used to create Non-Player Characters that should usually be used with a [Dialog Trigger](#_Dialog_Triggers).

The NPC component includes the following properties:

* **Character State:** The current character state. Used for testing.
* **Character Name:** The character’s name. Visible in the dialog and shop menus.
* **Image:** The character’s image. Visible in the dialog and shop menus.
* **Flip Image:** If checked, the character’s image will be flipped on the X-axis.
* **Standing:** Player’s standing with this character. **The standing system is not complete, and therefore this has no effect.**
* **Shop Contents:** A list of [Item](#_Item) objects that are sold by the character. **Note:** Opening the shop requires a [Dialog Trigger](#_Dialog_Triggers) with a dialog option that opens a shop ([SHOP\_OPEN](#SHOP_OPEN)).

## Location

The location component can be used as an apartment or a workplace. It can also be combined with a [Dialog Trigger](#_Dialog_Triggers) and/or an [NPC](#_NPC) component to create more complex interactions, e.g. have a store that is open from 8 am to 10pm.

The locations component contains the following properties:

* **Location Name:** The name of the location.
* **Address:** The address of the location (visible e.g., in job applications)
* **Rentable:** If checked, the player can rent the location. Doing this allows the player to sleep at this location.
* **Rent:** The amount of rent per 3 days. (TODO: Add rent interval to GameManager settings)
* **Workplace:** If checked, the player can work at this location.
* **ProfessionSO:** The profession object.
* **Special Location Type:** If set, automatically creates specialized dialog for the location. Most likely useless for any other project.
* **Open From:** The in-game time the location opens. The location can only be interacted with while open.
* **Open Until:** The in-game time the location closes.
* **Sleep From:** If set, determines the start of the range in which the player can sleep in this location.
* **Sleep Until:** If set, determines the end of the range in which the player can sleep in this location.
* **Open On:** The weekdays the location is open and can be interacted with.
* **Location Closed Popup:** The [Popup Container](#_Popup_Container) object that is triggered when the location is closed, and the player tries to interact with it.
* **Sleep Resource Change Per Hour:** The resources per hour the player gains and/or loses when sleeping in this location.
* **Sleep Random Events:** A selection of random events that randomly trigger when the player sleeps in this location.
* **Do Not Create Interaction Mark:** If checked, an interaction mark will not be automatically created to mark the location.

## Proximity Highlight

A component that is used to automatically highlight a GameObject in-game based on the player’s proximity to the object. **Important: Requires the GameObject to contain a Sprite Renderer** **component** **with a sprite reference.**

Proximity highlight is activated when the player is within the Proximity Highlight Distance. Interaction highlight is activated when pressing the interact button would cause the player to interact with the highlighted object.

The Proximity Highlight component contains the following properties:

* **Highlight Children:** If checked all children with a Sprite Renderer component will be highlighted as well.
* **Proximity Highlight Distance:** The distance from the player, at which the proximity highlight activates. 1 unit of distance is the change of one unit on the X or Y axis, in [world space](#_World_Space).
* **Proximity Highlight SO:** A [highlight](#_Highlight) object reference that determines the proximity highlight’s colors, thickness, etc.
* **Interaction Highlight SO:** A [highlight](#_Highlight) object reference that determines the interaction highlight’s colors, thickness, etc.

## Rotate

A component used to constantly rotate a GameObject at a specified speed. Updates on every frame. The speed is given as a Vector3, so per axis.

## Tooltip

The tooltip component can be added to any UI element. When hovering the cursor over the parent object, it automatically shows a tooltip element. When creating a tooltip, you can use one of the following properties:

* **Tooltip Text:** Sets a constant text for the tooltip. Good for elements that do not need a dynamic tooltip.
* **Tooltip Target ID:** The ID of the target object. Works only with objects that have a ToString() method implemented to them. The tooltip displays the text returned by ToString(). The only scriptable objects that support this component are [Trait](#_Trait) and [Item](#_Item).

## Street Light

This component works with [Lighting Manager](#_Lighting_Manager). If set up correctly, lighting manager turns streetlights on and off automatically.

## Light Flicker

Used to make a flickering light. Requires a Light2D component.

Contains the following properties:

* **Change Frequency:** Determines how often a change is made to the light’s properties. A range of seconds, chosen randomly between min and max. E.g., min = 1.5, max = 5: A number is randomly chosen between 1.5 and 5. Then after that number of seconds has passed, the light is updated with new values.
  + **Min:** The minimum number of seconds before changes are made to the light.
  + **Max:** The maximum number of seconds before changes are made to the light.
* **Use Inner Radius:** If checked, the inner radius of the light is changed randomly every time the light is updated.
* **Inner Radius:** Determines the range of the light’s inner radius.
  + **Min:** The minimum value for the light’s inner radius.
  + **Max:** The maximum value for the light’s inner radius.
* **Use Outer Radius:** If checked, the outer radius of the light is changed randomly every time the light is updated.
  + **Min:** The minimum value for the light’s outer radius.
  + **Max:** The maximum value for the light’s outer radius.
* **Use Intensity:** If checked, the intensity of the light is changed randomly every time the light is updated.
* **Intensity:** Determines the range of the light’s intensity.
  + **Min:** The minimum value for the light’s intensity.
  + **Max:** The maximum value for the light’s intensity.
* **Use Colors:** If checked, the color of the light is changed randomly every time the light is updated.
* **Color:** Determines the range of the light’s color.
  + **Start Color:** The start value for the light’s color.
  + **End Color:** The end value for the light’s color.

# Custom UI Components

## Flexible Grid Layout Group

This component can reposition and resize the UI element it is attached to, its parent and its children. This component is most useful if the children of a container want to be aligned to a row, column or a grid, that is relatively easily editable.

The component contains the following properties:

* **Padding:** The padding between the edges and child elements. **Note:** Only affects the edges. Spacing (below) is used for the gaps between children.
* **Child Alignment:** Determines the placement of the child objects. Default: Upper Left.
* **Ignore Sibling Height:** If checked, this object is placed in a way that ignores its siblings’ heights.
* **Constraint Type:** The type of constraint used for the grid alignment.
  + **Flexible:** The child objects are repositioned next to each other if they fit. If they don’t fit the last one not to fit will be moved to the next row.
  + **Fixed Column Count:** The grid will have a fixed number of columns. E.g., If Constraint is set to 1, the grid will be a one column “list”.
  + **Fixed Row Count:** The grid will have a fixed number of rows.
* **Constraint:** The numerical value of the constraint determined in Constrain Type.
* **Spacing:** Determines the length of the gap between child objects.
* **Transform Resizing:** This can be used to resize other objects that would not normally be affected by this component.
  + **Fit This:** The target object. This will be resized and repositioned.
  + **To This:** The reference. This object’s size and position will be used.
  + **Axis:** Determines on which axes the changes will be applied.
* **Do Not Reposition Parent Rect:** If checked, the parent of this object will not be automatically repositioned.
* **Do Not Reposition This Rect X:** If checked, this object will not be automatically repositioned on the X-axis.
* **Do Not Reposition This Rect Y:** If checked, this object will not be automatically repositioned on the Y-axis.
* **Move This Next To Target:** Used to move [this object](#_This) next to the target.
* **Target Side:** If Move This Next To Target (above) is set, this determines the side this object will be moved to.
* **Preserve Parent Position:** If checked, preserves the parent’s position if contents’ size change.
* **Preserve Child Aspect:** If checked, preserves the children’s aspect ratio when resizing them.
* **Fit Children To This:** If checked, fits children inside [this object](#_This).
* **Fit This To Children:** If checked, this object will be resized to fit the children’s size.
* **Fit Parent To This:** If checked, the parent will be fit to [this object’s](#_This) size.
* **Fit This To Parent:** If checked, [this object](#_This) will be resized to fit to the parent.
* **Call First:** A Unity Action that is called before resizing and repositioning any elements.
* **Call After Done:** A Unity Action that is called after all elements have been resized and repositioned.

## Game Action Click Trigger

This component is used to trigger a [Game Action](#_Game_Action_1) when the parent game object is clicked.

## Open Link On Click

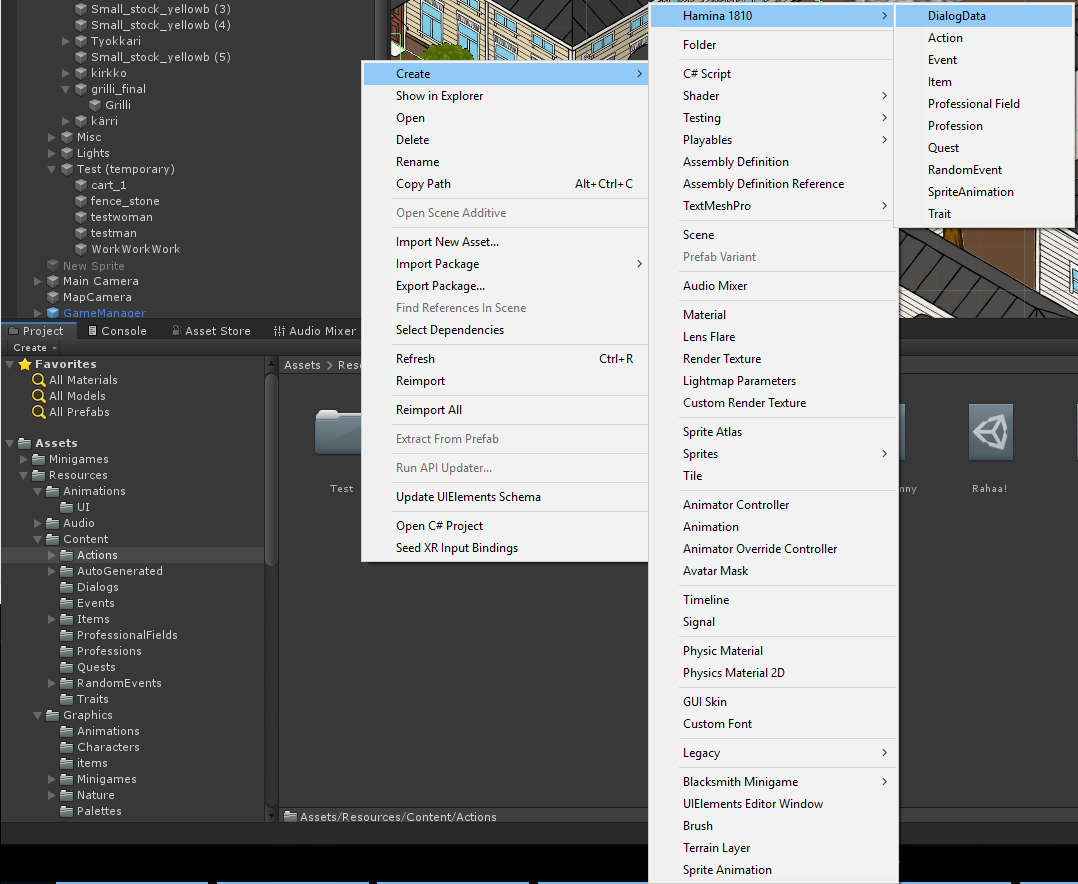
This component is used to open the specified URL in the device’s default browser application when the player clicks the object the component is attached to.

# Scriptable Objects

Scriptable objects are used to create persistent objects that require no programming from the content creator. Scriptable objects are automatically loaded from their respective folders. **Important:** Every scriptable object is supposed to have its own unique ID – unique to the type of object. These are generated automatically (if the ID = 0) when the game is launched in the editor. The ID cannot be changed manually.

**Warning:** If two objects somehow have the same ID, the system automatically finds a new ID for one of them and the references (e.g., in [Game Actions](#_Game_Action)) will break.

All different types of scriptable objects have their own folders under ‘Hamina 1810/Assets/Resources/Content/’. Feel free to organize your scriptable objects in subfolders within their own folders but make sure Events go under the Events folder and so on.

All different types of scriptable objects can be created by right-clicking in the Project tab/window and selecting one from ‘Create > Hamina 1810’ menu.

Remember to name your scriptable objects in a way that they’re easily recognized.

## Application Background

Stored in ‘Hamina 1810/Assets/Resources/Content/ApplicationBackgrounds’.

Used to create a changeable job application background. After setting up the scriptable object, it must be added to the ‘JobApplication\_Canvas’ prefab by adding a button as a child of the ‘BackgroundButtons\_Panel’ object, adding the ‘Change Background’ component to the button and finally adding a reference to the new scriptable object in the ‘Change Background’ component.

Application Background contains the following properties:

* **Background Image:** A job application background sprite.
* **Thumbnail Image:** A small sprite of the background to use as a thumbnail in the button.

## CV

Stored in ‘Hamina 1810/Assets/Resources/Content/CV’.

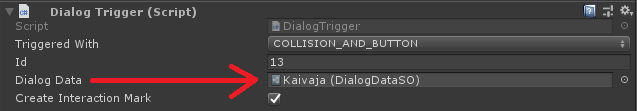
Used to create an editable CV page template.

CV contains the following properties:

* **Text:** The CV text. Changeable [keywords](#_Keywords) are added with index numbers starting from zero, inside curly braces. E.g., Text: “I am {0}.” If element 0 of the Keyword Types is set to PLAYER\_DESCRIPTION, the player can select and replace the “{0}” with one collected Player Description keyword.
* **Keyword Types:** An array of [keyword](#_Keywords) types. Select a correct type for each keyword index in the text. Element 0 replaces “{0}” in the text, elements 1 replaces “{1}” and so on.
  + **NONE:** Default value. Used to indicate a keyword type that hasn’t been set. Should be changed to something else.
  + **GREETING:** A changeable [greeting](#_Greeting) keyword type.
  + **VALEDICTION:** A changeable [valediction](#_Valediction) keyword type.
  + **JOB\_DESCRIPTION:** A changeable [job description](#_Job_Description) keyword type.
  + **EMPLOYER\_DESCRIPTION:** A changeable [employer description](#_Employer_Description) keyword type.
  + **MOTTO:** A changeable [motto](#_Motto) keyword type.
  + **CLASS:** The class of the player. Automatically retrieved from player data.
  + **PLAYER\_DESCRIPTION:** A changeable [player description](#_Player_Description) keyword type.
  + **BACKGROUND\_STORY:** A changeable [background story](#_Background_Story) keyword type.
  + **PLAYER\_GOAL:** A changeable player [goal keyword](#_Player_Goal) type.
  + **ADJECTIVE:** A player [trait](#_Trait) that is not marked as a main or a secondary skill. These are automatically retrieved from player data.
  + **MAIN\_SKILL:** A player [trait](#_Trait) that is marked as a main skill. These are automatically retrieved from player data.
  + **SECONDARY\_SKILL:** A player [trait](#_Trait) that is marked as a secondary skill. These are automatically retrieved from player data.
  + **FILLER\_PHRASE:** A changeable [filler phrase](#_Filler_Phrase) keyword type.
  + **PLAYER\_HAS\_EXPERIENCE:** Should only be used in job applications! A non-editable field that is automatically filled based on whether the player has gained experience in the professional field of the job that they are applying to.
  + **REFERENCE:** A non-editable field of [application references](#_Application_Reference) that is automatically retrieved from player data.
  + **PLAYER\_NAME:** A non-editable field that displays the player’s name.
* **Job Application SO:** A [job application](#_Job_Application) template that is connected to this CV object.

## Dialog Data

Stored in ‘Hamina 1810/Assets/Resources/Content/DialogData’.

Dialog Data objects are used to generate dialog within the game. They are most often linked to a Dialog Trigger. This can be done by dragging and dropping from the Project tab to the Inspector (image below):

Dialog Data contains the following properties:

* **Dialog Trigger ID:** Used to identify the [Dialog Trigger](#_Dialog_Triggers) used to trigger this dialog. This is automatically assigned if the object is linked to a Dialog Trigger.
* **Background Image:** The background image that will be shown behind the dialog view when one of this object’s dialogs is open. This is overridden if the active [Dialog](#_Game_Action) object has a background image.
* **NPC Name:** The NPC name shown in the dialog window.
* **NPC Image:** The NPC image shown in the dialog window.
* **Flip Image:** Flips the selected NPC Image on the X-axis.
* **Greetings:** An array of greetings. If none are found, a random one will be chosen from the default greetings. If more than one is found, one will be chosen randomly. The greeting is the first thing that appears in the dialog window.
* **Dynamic Greeting:** An array of [dynamic text](#_Dynamic_Text) greetings. If left empty, falls back to the normal greetings.
* **DialogSOs:** The actual [dialogs](#_Dialog) contained in the Dialog Data object.

## Dialog

Stored in ‘Hamina 1810/Assets/Resources/Content/Dialogs’.

Dialog objects are mostly referenced in [Dialog Data](#_Dialog_Data_1) objects.

Dialog contains the following properties:

* **Starts Active:** Determines if the dialog is automatically listed as one of the options when the dialog window is opened. Can be left unchecked and later activated with a [Game Action](#_Game_Action).
* **Background Image:** The background image that will be shown behind the dialog view when this dialog is open. Overrides the [Dialog Data](#_Dialog_Data_1) background image.
* **A screenshot of a video game

  Description automatically generatedStart Text:** The text that is shown on the button that starts the dialog in question.
* **Dynamic Start Text:** [Dynamic text](#_Dynamic_Text) that overrides the normal Start Text unless left empty.
* **Sentences:** An array of sentences the NPC will say after starting the dialog. If there are more than one, they’re shown in sequence.
* **Dynamic Sentences:** [Dynamic text](#_Dynamic_Text) that overrides the normal Sentences unless left empty.
* **Dialog Button Data:** An array of buttons which will be visible after all sentences have been shown.
  + **Button Text:** The text that is shown on the dialog button.
  + **Dynamic Button Text:** [Dynamic text](#_Dynamic_Text) that that overrides the normal Button Text unless left empty.

* + **[Prerequisites](#_Prerequisites):** Requirements that are used to check if the player can click the dialog button. If any of the requirements are not met, the button will be inactive.
  + **Button Action SOs:** An array of [Game Actions](#_Game_Action_1) containing all the actions that are executed when the button is pressed.

## Education

Stored in ‘Hamina 1810/Assets/Resources/Content/Educations’.

An Education contains the following properties:

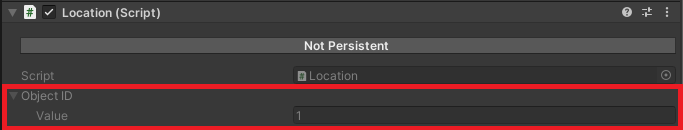
* **Name:** The name of the education.
* **Required Visits:** Number of visits required to complete the education.
* **Dialog Wait Text:** The text that will be shown in the dialog when the player can’t study.
* **Study Action:** The [GameAction](#_Game_Action_1) that triggers every time the player studies.
* **Finish Action:** The [GameAction](#_Game_Action_1) that triggers when the player has studied a number of times equal to **Required Visits**.
* **Add To CV:** If checked, the education is automatically added to the player’s CV.

## Game Action

Stored in ‘Hamina 1810/Assets/Resources/Content/GameActions’.

Game Actions are used to execute different kinds of actions within the game. The actions can start a dialog, change the in-game time, open an in-game popup window, move an object, etc.

A GameAction contains the following properties:

* **Ends Dialog:** Close the dialog window if it's open when the action is executed.
* **Add Or Remove Trait:** Adds or removes a trait from the player.
  + **Trait:** A trait scriptable object ([TraitSO](#_Trait)).
  + **Add:** If checked, adds the trait. If not checked, removes the trait.
* **Change Stat:** Change player stat.
  + **Stat:** The stat to be modified.
  + **Change:** The numeric value of the change. Positive numbers raise and negative number lower the stat.
* **Change Current Resource:** Changes the current player resource value. E.g., The player has 60/100 health points - the current health resource is the number 60. Then if **Min** is selected and **Change** is set to 20: the player's health points will be set to 20. If **Min** is NOT selected and **Change** is set to 20: the player will gain 20 health points which takes them to 80 health points.
  + **Type:** The resource type (Health, Sanity, etc.) to be modified.
  + **Min:** If checked, lowers the resource to its minimum value.
  + **Max:** If checked, raises the resource to its maximum value.
  + **Change:** The numeric value of the change. Positive numbers raise and negative number lower the resource amount.
* **Change Max Resource:** Changes the maximum player resource value. E.g., The player has 60/100 health points - the maximum health resource is the number 100.
  + **Type:** The resource type (Health, Sanity, etc.) to be modified.
  + **Integer Change:** Change in whole numbers. Positive numbers raise and negative numbers lower the maximum resource amount.
  + **Multiplier Change:** A decimal the maximum resource will be multiplied with. E.g., The player has 100 max health and **Multiplier Change** is set to 0.2: 100 \* 0.2 = 20 -> The player's max HP will be lowered to 20.
* **Change Standing (WIP): Currently not implemented.**
* **Action:** A list of miscellaneous actions.
  + **Action Name:** Name of the action. This is completely optional and is only used in a couple of very specific situations like debt collection.
  + **Action:** The type of the action to be executed.
    - **QUEST\_GIVE:** Starts a quest if the [prerequisites](#Prerequisites) are met. Automatically add the quest to the quest log under the [category](#_Quest_Category) specified in the quest object.
      * **Quest SO:** A [QuestSO](#_Quest) object.
    - **QUEST\_FINISH:** Finish a quest moving it from active to finished quests and executing the quest's finishing actions.
      * **QuestSO:** A [QuestSO](#_Quest) object.
    - **QUEST\_ADD\_LOG\_ENTRY:** Adds a quest log entry to the quest. These entries can be read from the quest log and are listed in the order they’re added.
      * **Text Area:** The text that will be added to the quest log.
    - **QUEST\_CHANGE\_CATEGORY:** Changes the [category](#_Quest_Category) a quest is listed under.
      * **Quest SO:** A [QuestSO](#_Quest) object.
      * **Quest Category SO:** The [category](#_Quest_Category) under which the quest will be moved.
    - **ITEM\_GIVE\_RANDOM:** Gives the player a random object from the loot table.
      * **Loot Table SO:** A [loot table](#_Loot_Table) object that determines the items and drop chances.
    - **ITEM\_GIVE:** Gives an item to the player.
      * **Item SO:** An [ItemSO](#_Item) object.
    - **ITEM\_REMOVE:** Removes an item from the player.
      * **Item SO:** An [ItemSO](#_Item) object.
    - **MONEY\_GIVE\_OR\_REMOVE:** Change the amount of money in the player's possession.
      * **Money Amount:** The amount of the change. A positive valueadds, and a negative value removes money.
    - **DEBT\_COLLECTION:** Triggers the debt collection event. **Name** is the name of the debt, e.g., "rent".
    - **CURSE\_REMOVE:** Removes curses from any cursed items equipped by the player.
    - **KEYWORD\_ADD:** Gives the player a keyword to use in their job applications and CV.
      * **Keyword Type:** Determines the type of the keyword.
      * **Scriptable Object (SO):** A scriptable object reference.
    - **KEYWORD\_REMOVE:** Removes a keyword from the player.
      * **Keyword Type:** Determines the type of the keyword.
      * **Scriptable Object (SO):** A scriptable object reference.
    - **DYNAMIC\_TEXT\_ADD\_CUSTOM:** Saves a custom text based on the selected type that can be later used in a [Dynamic Text](#_Dynamic_Text).
      * **Text Name:** The name of the text. This is later used to find the saved text. **Important: Does not tolerate errors! Case sensitive.**
      * **Text Type:** Determines the content of the saved text.
        + **NONE:** Default value. Saves nothing.
        + **PLAYER\_ADDRESS:** Saves player’s current home address.
        + **CURRENT\_SYSTEM\_TIME:** Saves current system time.
        + **CURRENT\_SYSTEM\_DATE:** Saves current system date.
        + **TOTAL\_TIME\_PLAYED:** Saves the total playtime of the current game.
        + **CURRENT\_GAME\_TIME:** Saves the current in-game time.
        + **CURRENT\_GAME\_DATE:** Saves the current in-game date.
        + **CURRENT\_SALARY:** Saves the player’s current salary.
    - **DIALOG\_ACTIVATE:** Activates a dialog so that it can be triggered by the player. Does NOT trigger the dialog.
      * **Dialog SO:** A [Dialog](#_Game_Action) object (**must be referenced to from a** [**DialogDataSO**](#_Dialog_Data_1) **object**).
    - **DIALOG\_DEACTIVATE:** Deactivates a dialog so that the player may no longer trigger it.
      * **Dialog SO:** A [dialog](#_Game_Action) object (**must be referenced to from a** [**DialogDataSO**](#_Dialog_Data_1) **object**).
    - **DIALOG\_START:** Triggers a [dialog](#_Game_Action). The dialog doesn't need to be active for this to work.
      * **Dialog SO:** A [dialog](#_Game_Action) object.
    - **DIALOG\_REACTION:** Shows an image in a speech bubble above the NPC’s character portrait.
      * **Reaction Image:** The sprite that will be inside the speech bubble.
      * **Reaction Time In Seconds:** The number seconds the reaction will be visible. If zero, the reaction disappears next time a dialog button is pressed.
    - **GAMEACTION\_TRIGGER:** Triggers another [GameAction](#_Game_Action_1).
      * **Game Action SO:** A [GameAction](#_Game_Action_1) object.
    - **JOB\_INTERVIEW\_START:** Starts a special job interview dialog for the specified profession and starts keeping track of the interview points based on the player’s dialog choices. Job interview dialog uses custom graphics specified in the [Dialog Manager](#_Dialog_Manager) component.
      * **Dialog SO:** A [dialog](#_Game_Action) object.
      * **Profession SO:** A [profession](#_Profession) object. The profession the player is applying for.
    - **JOB\_INTERVIEW\_END:** Ends the interview and triggers either the positive or the negative [GameAction](#_Game_Action_1) based on the [profession’s](#_Profession) properties and the points accumulated during the interview.
    - **JOB\_INTERVIEW\_CHANGE\_SCORE:** Used in job interview dialogs to keep track of how well the interview is going. The total number of points determines the interview outcome.
      * **Interview Score Change:** The number of points gained or lost.
    - **EVENT\_START:** Adds an event to the active events list. Does not instantly trigger the event, only starts the countdown towards it.
      * **Game Event SO:** A [GameEventSO](#_Game_Event) object.
    - **POPUP\_SHOW:** Shows a popup based on the properties of the popup object.
      * **Popup Container SO:** A [popup](#_Popup_Container) object.
    - **TRIGGER\_RANDOM\_EVENT:** Randomly triggers an event from a [table of random events](#_Random_Event_Trigger).
      * **Random Event Trigger Table SO:** A [Random Event Trigger Table](#_Random_Event_Trigger) object.
    - **CHANGE\_TABLE\_TRIGGER\_CHANCE:** Changes the trigger chance of an item within a [Random Event Trigger Table](#_Random_Event_Trigger).
      * **Random Event Trigger Table SO:** The trigger table that contains the item.
      * **Random Event Trigger Table Item SO:** The [item](#_Random_Event_Trigger_1) whose trigger weight will be changed.
      * **New Chance:** The new trigger weight that will be assigned to the referenced item found in the referenced table.
    - **CHANGE\_WORK\_TABLE\_TRIGGER\_CHANCE:** Changes the trigger chance of an item within a [Random Work Event Table](#_Random_Work_Event).
      * **Random Work Event Table SO:** The [table](#_Random_Work_Event) that contains the item.
      * **Random Work Event Table Item SO:** The [item](#_Random_Work_Event_1) whose trigger weight will be changed.
      * **New Chance:** The new trigger weight that will be assigned to the referenced item found in the referenced table.
    - **JOB\_START:** Changes the player's profession.
      * **Profession SO:** A [ProfessionSO](#_Profession) object.
    - **JOB\_QUIT:** Quits the specified job or any job.
      * **Profession SO:** A [ProfessionSO](#_Profession) object. If specified, quits that specific job. If left empty, quits any job.
    - **JOB\_APPLY:** Opens the job application view for the specified job.
      * **Profession SO:** A [profession](#_Profession) object. The target job.
    - **JOB\_LOCATION:** Starts tracking a job's location.
      * **Profession SO:** A [ProfessionSO](#_Profession) object.
    - **WORK:** Attempts to start a workday.
      * **Profession SO:** The target [profession](#_Profession).
    - **PROFESSION\_CHANGE\_POINTS:** Changes Profession Points (kind of a tracker of how well the player is doing in their job). Points can trigger [GameActions](#_Game_Action_1) when in a certain range, specified in the [profession’s](#_Profession) Point Actions.
      * **Profession SO:** The target [profession](#_Profession).
      * **Change:** The number of points gained or lost.
    - **CHANGE\_CURRENT\_JOB\_PAY\_MULTIPLIER:** Changes the salary multiplier of the player’s current job.
      * **New Multiplier:** The number the player’s salary will be multiplied by before getting paid. E.g., 0.5 = 50% of normal pay and 2.5 = 250%.
    - **CHANGE\_CV:** Changes the active CV of the player.
      * **CV SO:** A [CV](#_CV) object.
    - **ADD\_APPLICATION\_BACKGROUND:** Gives the player access to the specified application background (when filling a job application).
      * **Application Background SO:** An [application background](#_Application_Background) object.
    - **EDUCATION\_ACTIVATE:** Activates the target education at the target [location](#_Location).
      * **Education SO:** An [Education](#_Education) object.
      * **Location ID:** The ID number of the [Location](#_Location) component where the studies take place.  
          
        The button under the Location ID will be red if an invalid ID is detected.
    - **EDUCATION\_START:** Optional step. Adds the starting time to the player’s CV, if ‘Add To CV’ is checked in the [education](#_Education) object. This action is completed automatically when EDUCATION\_STUDY is called for the first time.
      * **Education SO:** The target [Education](#_Education) object.
    - **EDUCATION\_FINISH:** Optional step. Triggers the ‘Finish Action’ specified in the [education](#_Education) object and adds the ending time to the player’s CV, if ‘Add To CV’ is checked in the [education](#_Education) object. This action is completed automatically when EDUCATION\_STUDY is called for the last time.
      * **Education SO:** The target [Education](#_Education) object.
    - **EDUCATION\_STUDY:** Triggers the ‘Study Action’ specified in the [education](#_Education) object. Automatically keeps count of the number of times the player has studied this subject.
      * **Education SO:** The target [Education](#_Education) object.
    - **SHOP\_OPEN:** Opens a shop.
      * **NPC ID:** Can be set to an ID of an NPC object to open a shop from anywhere or left empty if the object itself has a shop and a [Location](#_Location) component.
    - **HOME\_CHANGE:** Attempts to change the player's home to the current [Location](#_Location).
      * **Location ID:** The ID number of the target [Location](#_Location). If zero, tries only the current location.
    - **SLEEP:** Opens a sleep popup window if ‘Use Sleep Popup’ is checked in the [GameManager](#_Game_Manager) component. If ‘Use Sleep Popup’ is not checked, player sleeps the amount specified. **NOTE:** This is also used automatically by the autogenerated dialog options.
      * **Hours:** The number of hours to sleep. Only visible/used if ‘Use Sleep Popup’ is unchecked in the [GameManager](#_Game_Manager) component.
    - **HOME\_PAY\_RENT:** Attempts to pay rent. **NOTE:** This is used automatically by the automated rent system and should NOT be used manually at its current state. (Maybe in the future)
    - **SET\_HOMELESS:** Sets the player homeless.
    - **TRACKER\_ARROW\_DISABLE:** Disable the tracker arrow.
    - **ARROW\_TRACK\_OBJECTBASEMONO:** Enable the tracker arrow and aim it at the target [ObjectBaseMono](#_Object_Base_Mono) object.
      * **Target Type:** The type of the [ObjectBaseMono](#_Object_Base_Mono) object.
        + **OBJECT\_BASE\_MONO:** An [ObjectBaseMono](#_Object_Base_Mono) object
        + **DIALOG\_TRIGGER:** A [Dialog Trigger](#_Dialog_Triggers) object
        + **LOCATION:** A [Location](#_Location) object
        + **NPC:** An [NPC](#_NPC) object
        + **CHARACTER:** A [Character](#_Character) object.
        + **PLAYER:** The [player](#_Player).
        + **NONE:** No target.
      * **Object Base Mono ID:** The ID number of the target object.
      * **Disable Distance:** The distance at which the arrow will be disabled.
    - **ARROW\_TRACK\_POSITION:** Enable the tracker arrow and aim it at a target position.
      * **Position:** The position where the arrow will be aimed at.
      * **Disable Distance:** The distance at which the arrow will be disabled.
      * **Show Position Handle:** If checked, a draggable position handle will be visible in the editor.
    - **GAMEOBJECT\_CHANGE\_SPRITE:** Change the sprite of a GameObject. **Important:** Requires the GameObject to contain a [Sprite Renderer](#_Sprite_Renderer) component.
      * **Target Type:** The type of the [ObjectBaseMono](#_Object_Base_Mono) object.
      * **Object Base Mono ID:** The ID number of the target object.
      * **Sprite:** The new sprite.
    - **HIGHLIGHT\_UI\_ELEMENT:** Highlights a UI element. **Important:** Requires the UI element to contain an [ObjectBaseMono](#_Object_Base_Mono) component.
      * **Object Base Mono ID:** The ID number of the target object.
      * **Highlight Length:** The total length of the highlight effect in seconds.
      * **Highlight Transition Time:** The transition time of the highlight between visible and invisible.
    - **HIGHLIGHT\_GAMEOBJECT\_START:** Highlights a GameObject.
      * **Target Type:** Target type.
        + **OBJECT\_BASE\_MONO:** An [ObjectBaseMono](#_Object_Base_Mono) object
        + **DIALOG\_TRIGGER:** A [Dialog Trigger](#_Dialog_Triggers) object
        + **LOCATION:** A [Location](#_Location) object
        + **NPC:** An [NPC](#_NPC) object
        + **CHARACTER:** A [Character](#_Character) object.
        + **PLAYER:** The [player](#_Player).
        + **NONE:** No target.
      * **Object Base Mono ID:** The ID number of the target object.
      * **Highlight SO:** A [Highlight](#_Highlight) object.
    - **HIGHLIGHT\_GAMEOBJECT\_STOP:** Removes a highlight effect from a GameObject.
      * **Target Type:** Same as **HIGHLIGHT\_GAMEOBJECT\_START** Target Type (above).
      * The ID number of the target object.
    - **PARTICLES\_START:** Activates a particle system based on the following properties:
      * **Particle Name:** The name of the particle system. Used later when turning off the particles. **Note:** Case sensitive!
      * **Parent Type:** Type of the parent (target) object:
        + **PLAYER:** The [player](#_Player).
        + **DIALOG\_TRIGGER:** A [Dialog Trigger](#_Dialog_Triggers) object.
        + **LOCATION:** A [Location](#_Location) object.
        + **NPC:** An [NPC](#_NPC) object.
        + **CHARACTER:** A [Character](#_Character) object.
        + **OBJECT\_BASE\_MONO:** An [ObjectBaseMono](#_Object_Base_Mono) object.
      * **Particle Material:** The material of the particles.
      * **Position:** The local position of the particle system.
      * **Particle Life Time:** The lifetime range of the particles in seconds.
        + **Min:** Minimum lifetime.
        + **Max:** Maximum lifetime.
      * **Particle Speed:** The speed range of the particles.
        + **Min:** Minimum speed.
        + **Max:** Maximum speed.
      * **Particles Per Second:** The range of the number of particles spawned per second.
        + **Min:** Minimum particles per second.
        + **Max:** Minimum particles per second.
      * **Particle Size:** Particle size range.
        + **Min:** Minimum particle size.
        + **Max:** Maximum particle size.
    - **PARTICLES\_STOP:** Stop a particle system.
      * **Particle Name:** The name of the particle system. **Note:** Case sensitive!
    - **OVERLAY\_PROPERTIES\_APPLY:** Applies the given overlay properties.
      * **Overlay Properties SO:** An [Overlay Properties](#_Overlay_Properties) object.
    - **CHANGE\_TIME\_MULTIPLIER:** Changes how quickly the time runs.
      * **Time Multiplier:** The new time multiplier. 1 = normal time, 0.5 = half speed, 2 = double speed.
    - **CHANGE\_GAME\_TIME:** Changes in-game time.
      * **Time Type:** Determines how the time will change.
        + **FROM\_NOW:** Changes the time relative to current time.
        + **TIME\_OF\_DAY:** Changes the game time to an absolute time.
        + **ABSOLUTE\_TIME\_RELATIVE\_DAY:** Changes the time to an absolute time but preserves the current day.
      * **Game Time:** The time.
    - **FADE\_CAMERA:** Draws and updates an overlay color on the camera, based on given properties.
      * **Fade State:** The direction of the fade.
        + **STOPPED:** Stops any current fade on the camera.
        + **IN:** Fade in, overlay goes from opaque to transparent.
        + **OUT:** Fade out, overlay goes from transparent to opaque.
      * **Fade Settings:**
        + **Speed Scale:** The scale multiplier of how long the fade lasts. By default, the fade lasts 1 second. If speed scale is set to 2, the fade lasts 0.5 seconds.
        + **Fade Color:** The color of the fade overlay.
        + **Fade Curve:** The curve of the fade. The higher the curve, the opaquer the color.
        + **Trigger After Fade:** A [GameActionSO](#_Game_Action_1) that will be triggered after the fade finishes.
    - **PLAY\_VIDEO:** Plays a full screen video on top of all other game elements.
      * **Video Source:** The source of the video.
        + **Video Clip:** A video clip located in the StreamingAssets folder.
        + **URL:** A URL pointing to a video file.
      * **Fade In Speed Scale:** The scale multiplier of how long the fade lasts. By default, the fade lasts 1 second. If speed scale is set to 2, the fade lasts 0.5 seconds.
      * **Fade Out Speed Scale:** The scale multiplier of how long the fade lasts. By default, the fade lasts 1 second. If speed scale is set to 2, the fade lasts 0.5 seconds.
      * **Fade Color:** The color of the fade overlay.
      * **Mute Music:** If selected, the music will be muted while the video is playing.
      * **Triggered After Video:** A [GameActionSO](#_Game_Action_1) that will be triggered after the video has finished playing.
      * **Wait Until Triggered:** Seconds to wait before triggering the [GameActionSO](#_Game_Action_1) (Triggered After Video).
    - **LOAD\_SCENE:** Loads a new scene.
      * **Scene Index:** The target scene build index. Found in the build settings (Shortcut key: Ctrl + Shift + B).
      * **Player Position:** The world position to where the player will be moved after loading the new scene.
    - **JOBCENTER\_JOB\_SEARCH:** Opens a dialog with all the job opportunities the player is suited for (passes all the requirements). **Value** is ignored.
    - **KELA\_REPORT:** A special action that creates the event that gives the player their daily allowance.
  + **Value:** Usually the ID of an object. For a more detailed explanation, see the action types above.
* **Activate Triggers:** An array of TriggerSOs to be activated when this Game Action is executed.
* **Move Object:** Moves a game object in the current scene.
  + **Target:** The target object type.
    - **THIS:** The object that is determined to be this object's owner. E.g., If the action is contained in a dialog the owner will the DialogTrigger and the object that will be moved will be the object containing said DialogTrigger.
    - **PLAYER:** The target will be the player.
    - **CAMERA:** The target will be the camera.
    - **DIALOG\_TRIGGER:** The target will be an object that contains a [DialogTrigger](#_Dialog_Triggers) component.
    - **LOCATION:** The target will be an object that contains a [Location](#_Location) component.
    - **NPC:** The target will be an object that contains an [NPC](#_NPC) component.
    - **CHARACTER:** The target will be an object that contains a [Character](#_Character) component.
    - **OBJECT\_BASE\_MONO:** The target will be an object that contains an [ObjectBaseMono](#_Object_Base_Mono) component.
  + **Mono Object ID:** The ID of the object to be moved. Visible if **Target** is set to **DIALOG\_TRIGGER**, **LOCATION**, **NPC**, **CHARACTER**, or **OBJECT\_BASE\_MONO**.
  + **Directions And Speeds:** The directions and speeds of the movements. Always executed in order from top to bottom.
    - **Movement Type:** The type of the movement.
      * **INSTANT:** Instantly moves the object.
      * **TRANSLATE:** Moves the object towards the target location at a speed dictated by **Translation Curve** and **Speed**.
    - **Translation Curve:** A curve used to calculate the speed based on the distance travelled. X-axis on the curve is the relative distance travelled and Y-axis is the speed multiplier. E.g., If the curve is at X = 0.5 and Y = 1: the object will move at full **Speed** when at the midway point of the movement. **NOTE:** There's a failsafe in the system that ignores a speed multiplier of 0 since this would cause the object to stop moving and the relative distance to the target staying the same, which would in turn cause the movement to stay in an infinite loop. So, if a value of 0 is found on the Y-axis the system automatically replaces it with a value of 0.01.
    - **Coordinate Type:** The type of coordinates used for the movement. Distances are based on Unity's coordinate system.
      * **DIRECTION:** The target location will be calculated from the object's own location. E.g., Object is located at X = 10, Y = 10 and **Coordinates** are set to X = -2, Y = 4: The target location will be X: 10 + -2 = 8, Y: 10 + 4 = 14.
      * **POSITION:** The target position in the Unity coordinate system.
    - **Coordinates:** The coordinates. See [Coordinate Type](#CoordinateType) above.
    - **Movement Length:** The length of this movement in seconds.
    - **Pause For Seconds:** Pause between the end of this movement and the start of the next one (in seconds).
  + **Loop:** If selected the movements will loop from the start.
  + **Loop Times:** Number of times the movements will be looped (0 = infinity).
* **Change In Game Time:** Change the in-game time. Positive values make time go forward and negative values make it go backwards.
  + **Days:** In-game days
  + **Hours:** In-game hours
  + **Minutes:** In-game minutes
  + **Seconds:** In-game seconds
* **Open Website:** Opens a pop-up window asking the player if they want to visit the **Web Address**. If the player selects yes, the **Web Address** will be opened in the default browser. If the player selects no, the pop-up is closed, and nothing happens.
  + **Web Address:** The address of the website.
* **Popup:** Opens a customizable pop-up window.
  + **Popup properties:** The properties of the pop-up window.
    - **Element Height:** The height of the pop-up window as a percentage of the screen height. E.g., A value of 0.2 would be 20% of the screen height.
    - **Element Width:** The width of the pop-up window as a percentage of the screen width. E.g., A value of 0.2 would be 20% of the screen width.
    - **Horizontal Alignment:** The horizontal alignment of the pop-up window.
    - **Margin Horizontal:** The margin that should be left between the pop-up and the screen edge, as a percentage of the screen width. E.g., A value of 0.2 would leave a gap of 20% of the screen width between the edge of the screen and the pop-up window. **NOTE:** Used only if **Horizontal Alignment** is set to **LEFT** or **RIGHT**. Setting a margin when the element is centered does nothing.
    - **Vertical Alignment:** The vertical alignment of the pop-up window.
    - **Margin Vertical:** The margin that should be left between the pop-up and the screen edge, as a percentage of the screen height. E.g., A value of 0.2 would leave a gap of 20% of the screen height between the edge of the screen and the pop-up window. **NOTE:** Used only if **Vertical Alignment** is set to **TOP** or **BOTTOM**. Setting a margin when the element is centered does nothing.
  + **Background Image:** The image that will cover the whole pop-up window.
  + **Background Color:** The color of the **Background Image**.
  + **Picture:** An image that will be shown on the pop-up.
  + **Picture Color:** The color of the **Picture**.
  + **Picture Properties (A.K.A. Element Properties):** The properties of the picture (or another pop-up element).
    - **Element Height:** The height of the element as a percentage of the parent height. E.g., A value of 0.2 would be 20% of the parent's height.
    - **Element Width:** The width of the element as a percentage of the parent width. E.g., A value of 0.2 would be 20% of the parent's width.
    - **Horizontal Alignment:** The horizontal alignment of the element within the parent's borders.
    - **Margin Horizontal:** The margin that should be left between the element and the parent's border, as a percentage of the parent's width. E.g., A value of 0.2 would leave a gap of 20% of the parent's width between the border of the parent element and this element. **NOTE:** Used only if **Horizontal Alignment** is set to **LEFT** or **RIGHT**. Setting a margin when the element is centered does nothing.
    - **Vertical Alignment:** The vertical alignment of the element within the parent's borders.
    - **Margin Vertical:** The margin that should be left between the element and the parent's border, as a percentage of the parent's height. E.g., A value of 0.2 would leave a gap of 20% of the parent's height between the border of the parent element and this element. **NOTE:** Used only if **Vertical Alignment** is set to **TOP** or **BOTTOM**. Setting a margin when the element is centered does nothing.
  + **Text Color:** The color of the **Popup Text**.
  + **Popup Text:** The text that will be shown on the pop-up window. **NOTE:** Currently this must have a value for the pop-up to show. If no text is wanted, set it to " " (white space).
  + **Text Properties:** The properties of the **Popup Text**, see [Element Properties](#ElementProperties).
  + **Close Button: NOTE:** Currently either the **Close Button** or a custom **Popup Button** is required.
  + **Close Button Properties:** The properties of the Close button, see [Element Properties](#ElementProperties).
  + **Ok Button Properties:** The properties of the OK button, see [Element Properties](#ElementProperties).
  + **Popup Buttons:** Customizable buttons that can have actions linked to them. **NOTE:** Currently either a custom **Popup Button** or the **Close Button** is required.
    - **Button Text:** The text that will be shown on the button.
    - **Button Properties:** The properties of the button, see [Element Properties](#ElementProperties).
    - **Game Actions:** The GameActionSOs which will be triggered with the button press, see [Game Action](#_Game_Action).
    - **Action IDs:** IDs of GameActions which will be triggered with the button press, see [Game Action](#_Game_Action). **NOTE:** Can be used to refer to game actions within other objects, like Dialogs.

## Game Event

Stored in ‘Hamina 1810/Assets/Resources/Content/Events’.

Game Events are timed Game Actions. They can be timed either relatively from the time they're activated or set to be triggered at a specific time.

* **Visible In Calendar:** Checked if the event is shown in the calendar. **NOTE:** Calendar is not yet implemented.
* **Activate Instantly:** Check if the event should be activated instantly. Activating an event does not trigger it but allows it to be triggered then the target time is reached.
* **Time Type:** The way the target time is calculated.
  + **FROM\_NOW:** The event execution time is relative from the moment it's activated. E.g., If **Time** is set to 2 days, 0 hours, 0 minutes, 0 seconds: the event will trigger 2 days from when it was activated.
  + **TIME\_OF\_DAY:** The event execution time is fixed. E.g., If **Time** is set to 2 days, 0 hours, 0 minutes, 0 seconds: the event will trigger at midnight between day 1 and day 2.
* **Time:** The time when the event **Actions** will be triggered. See [Time Type](#TimeType) for more information about relative and fixed times.
* **Actions:** The Game Actions that will be triggered at the event execution time. See [Game Action](#_Game_Action).

## Highlight

Stored in ‘Hamina 1810/Assets/Resources/Content/Effects/Highlights’.

A person in a garment

Description automatically generated with low confidenceA Scriptable Object used in the [Proximity Highlight](#_Proximity_Highlight) component to highlight game objects in the scene. The highlight color is blended from the start color and end color, based on the timer’s relative position on a sine wave.

Simply put: the highlight starts from the start color and works its way towards the end color, and then back to the start color, and so on.

* **Thickness:** Thickness of the highlight. This is dependent on the size of the target sprite (in the Sprite Renderer component).
* **Speed:** The speed at which the highlight color changes from start color to end color.
* **Start Color:** The start color of the highlight.
* **End Color:** The end color of the highlight.

## Item

Stored in ‘Hamina 1810/Assets/Resources/Content/Items’.

* **Icon:** The picture representing the object in the game UI.
* **Name:** Name of the object.
* **Description:** The item description.
* **Price:** The price of the item.
* **Indestructible:** If checked, the item cannot be destroyed by the player.
* **Not Sellable:** If checked, the item cannot be sold by the player.
* **Cursed:** If checked, the item will be cursed and cannot be unequipped unless the curse is removed.
* **Stackable:** If checked, the item stacks in one inventory slot.
* **Equippable:** If checked, the item can be equipped by the player.
* **Equipped In Slot:** The equipment slots to which the item can be equipped.
* **Equip Effects:** Effects that are triggered when the item is equipped.
  + **Effects:** The [Game Action](#_Game_Action) to be triggered when the item is equipped.
  + **Effects Last:** The time the **Effects** last. If set to 0, the effects last forever.
* **Usable:** If checked, the item can be used. Using an item doesn’t consume the item.
* **Use Effects:** Effects that are triggered when the item is used.
  + **Effects:** The [Game Action](#_Game_Action) to be triggered when the item is used.
  + **Effects Last:** The time the **Effects** last. If set to 0, the effects last forever.
* **Consumable:** If checked, the item can be consumed. Consuming an item is like using it, but it destroys the item.
* **Consume Effects:** Effects that are triggered when the item is consumed.
  + **Effects:** The [Game Action](#_Game_Action) to be triggered when the item is consumed.
  + **Effects Last:** The time the **Effects** last. If set to 0, the effects last forever.
* **On Pickup:** A list of [Game Action](#_Game_Action)s that are triggered when the item is acquired by the player.
* **On Remove:** A list of [Game Action](#_Game_Action)s that are triggered when the player destroys, sells or otherwise loses the item.

## Job Application

Stored in ‘Hamina 1810/Assets/Resources/Content/JobApplications.

The JobApplicationSO is used add editable text to the job application UI.

* **Text:** The text that will be placed in the job application text area. The keyword types will be automatically filled in any number surrounded by curly brackets. For example, {1} points to the second element (Element 1) in the Keyword Types. **WARNING:** The game will break if the tag in the text does not point to an element in the Keyword Types array.
* **Keyword Types:** An array of keyword types that correspond to the tags mentioned above.
  + **NONE:** Does nothing.
  + **GREETING:** Selectable from a list of available [greetings](#_Greeting).
  + **VALEDICTION:** Selectable from a list of available [valedictions](#_Valediction).
  + **JOB\_DESCRIPTION:** Selectable from a list of available [job descriptions](#_Job_Description).
  + **EMPLOYER\_DESCRIPTION:** Selectable from a list of available [employer descriptions](#_Employer_Description).
  + **MOTTO:** Selectable from a list of available [mottos](#_Motto).
  + **CLASS:** Nonselectable. Player’s class, determined by the selected [tarot card](#_Tarot_Card) when selecting a character.
  + **PLAYER\_DESCRIPTION:** Selectable from a list of available [player descriptions](#_Player_Description).
  + **BACKGROUND\_STORY:** Selectable from a list of available [background stories](#_Background_Story).
  + **PLAYER\_GOAL:** Selectable from a list of available [player goals](#_Player_Goal).
  + **ADJECTIVE:** Selectable from a list of available adjectives. Available adjectives are acquired player [traits](#_Trait) whose skill type is neither MAIN nor SECONDARY.
  + **MAIN\_SKILL:** Selectable from a list of available main skills. Available main skills are acquired player [traits](#_Trait) whose skill type is MAIN.
  + **SECONDARY\_SKILL:** Selectable from a list of available secondary skills. Available secondary skills are acquired player [traits](#_Trait) whose skill type is SECONDARY.
  + **FILLER\_PHRASE:** Selectable from a list of available [filler phrases](#_Filler_Phrase).
  + **PLAYER\_HAS\_EXPERIENCE:** Nonselectable. Automatically set to either “on” (Eng. has) if the player has experience in the current [profession’s](#_Profession) [field](#_Professional_Field) or “ei ole” (Eng. doesn’t have) if they don’t have experience.
  + **REFERENCE:** Nonselectable. All acquired [application references](#_Application_Reference) are listed automatically.
  + **PLAYER\_NAME:** Nonselectable. The player’s name is displayed automatically.

## Keywords

Stored in ‘Hamina 1810/Assets/Resources/Content/Keywords’.

Keywords are [ScriptableObjects](#_Scriptable_Objects) used mainly in the [CV](#_CV) and [Job Application](#_Job_Application) objects. They’re used to select a type of a word to fill a slot in an editable text. The player must have gotten the keyword previously (KEYWORD\_ADD in the [Game Action](#_Game_Action_1)) to be able to select it in the menu.

There are two types of keywords: selectable and nonselectable. The selectable type can be selected from a popup menu and the nonselectable type will be automatically filled based on acquired keywords or other parameters.

All ScriptableObject type keywords contain the same properties. The context menu text option is ignored for Application Reference type since it’s a nonselectable keyword. Keywords include the following properties:

* **Text:** The text that will be shown on the job application or CV.
* **Context Menu Text:** The text that will be shown in the context menu for this item.
* **Global Weight:** The global value of the keyword. Used if neither the [professional field](#_Professional_Field) nor the [profession](#_Profession) have a weight set for this keyword.

### Application Reference

A nonselectable keyword that represents a reference from a previous employer. All acquired references will be listed automatically if used in the [job application](#_Job_Application) or [CV](#_CV).

### Background Story

A selectable keyword for a player background story.

### Employer Description

A selectable keyword for an employer description.

### Filler Phrase

A selectable keyword for a filler phrase.

### Greeting

A selectable keyword for a greeting.

### Job Description

A selectable keyword for a job description.

### Motto

A selectable keyword for a motto.

### Player Description

A selectable keyword for a player description.

### Player Goal

A selectable keyword for a player goal.

### Valediction

A selectable keyword for a valediction.

## Loot Table

Stored in ‘Hamina 1810/Assets/Resources/Content/LootTables’.

The loot table enable giving the player a random item from the array of item contained within. Loot tables are used with the [ITEM\_GIVE\_RANDOM](#ITEM_GIVE_RANDOM) [Game Action](#_Game_Action_1).

* **Items:** An array of items and loot chances.
  + **Item:** The item. Can be left empty to have a chance of giving no loot to the player.
  + **Loot Chance:** The weight which determines the chance at which the item will be randomly chosen. **Example**: If this item has a weight of 100 and the rest of the item in the table have a combined chance of 900. In this case this item would have a chance of 10% to be randomly chosen. This scenario would work the same way with the numbers 10 and 90, and 1 and 9.

## Overlay Properties

Stored in ‘Hamina 1810/Assets/Resources/Content/OverlayProperties’.

Overlay properties objects are used to display an effect (like fog or rain) over the game world. Making new effects requires some knowledge on how to create or modify Unity’s Shader Graphs.

Overlay Properties contains the following properties:

* **Visible:** Determines if the effect is visible or not.
* **Overlay Material:** The material of the effect. Determines the texture and the shader used for the effect.
* **Overlay Speed:** The speed and direction at which the texture is moved.
* **Overlay Size:** Basically, works like a zoom. The bigger the number, the further away the texture seems to be.
* **Overlay Color:** A color that will be blended to the texture.

## Tiled Overlay Properties

Stored in ‘Hamina 1810/Assets/Resources/Content/OverlayProperties’.

The same as the Overlay Properties but allows texture tiling.

* **Overlay Tiling:** Allows stretching of the texture on both the X and the Y axes.

## Popup Container

Stored in ‘Hamina 1810/Assets/Resources/Content/Popups’.

Opens a popup based on the given parameters. Used with the [POPUP\_SHOW](#POPUP_SHOW) [Game Action](#_Game_Action_1).

Popup Container contains the following properties:

* **Popup Type:** The type of the popup. Determines the visible elements in the popup window, as well as part the functionality.
  + **BOOLEAN:** A basic popup with buttons for yes and no answers (OK and cancel).
  + **CUSTOM:** A fully customizable popup.
  + **SLIDER:** A popup with a slider.
  + **WORK:** A popup for selecting workday length.
  + **SLEEP:** A popup for selecting sleep length.
* **Popup properties (or Element Properties):** The properties of the pop-up window.
  + **Element Height:** The height of the pop-up window as a percentage of the screen height. E.g., A value of 0.2 would be 20% of the screen height.
  + **Element Width:** The width of the pop-up window as a percentage of the screen width. E.g., A value of 0.2 would be 20% of the screen width.
  + **Horizontal Alignment:** The horizontal alignment of the pop-up window.
  + **Margin Horizontal:** The margin that should be left between the pop-up and the screen edge, as a percentage of the screen width. E.g., A value of 0.2 would leave a gap of 20% of the screen width between the edge of the screen and the pop-up window. **NOTE:** Used only if **Horizontal Alignment** is set to **LEFT** or **RIGHT**. Setting a margin when the element is centered does nothing.
  + **Vertical Alignment:** The vertical alignment of the pop-up window.
  + **Margin Vertical:** The margin that should be left between the pop-up and the screen edge, as a percentage of the screen height. E.g., A value of 0.2 would leave a gap of 20% of the screen height between the edge of the screen and the pop-up window. **NOTE:** Used only if **Vertical Alignment** is set to **TOP** or **BOTTOM**. Setting a margin when the element is centered does nothing.
* **Background Image:** The image that will cover the whole pop-up window.
* **Background Color:** The color of the **Background Image**.
* **Video Clip:** A video clip.
* **Video Url:** A URL to a video file location.
* **Video Properties:** The properties (see [Popup Properties](#ElementProperties)) of the video player element.
* **Pictures:** An array of pictures.
  + **Picture:** An image that will be shown on the pop-up.
  + **Picture Color:** The color of the **Picture**.
  + **Picture Properties:** The properties (see [Popup Properties](#ElementProperties)) of the picture.
* **Text Color:** The color of the **Popup Text**.
* **Popup Text:** The text that will be shown on the pop-up window. **NOTE:** Currently this must have a value for the pop-up to show. If no text is wanted, set it to " " (white space).
* **Dynamic Text:** Can be used instead of Popup Text. For instructions on how to use it, see [Dynamic Text](#_Dynamic_Text).
* **Text Properties:** The properties of the **Popup Text** or **Dynamic Text**, see [Element Properties](#ElementProperties).
* **Close Button: NOTE:** Currently either the **Close Button** or a custom **Popup Button** is required.
* **Close Button Properties:** The properties of the Close button, see [Element Properties](#ElementProperties).
* **Ok Button:** If selected an OK button will be visible.
* **Ok Button Properties:** The properties of the OK button, see [Element Properties](#ElementProperties).
* **Popup Buttons:** Customizable buttons that can have actions linked to them. **NOTE:** Currently either a custom **Popup Button** or the **Close Button** is required.
  + **Button Text:** The text that will be shown on the button.
  + **Dynamic Button Text:** The [dynamic text](#_Dynamic_Text) that will be shown on the button.
  + **Button Properties:** The properties of the button, see [Element Properties](#ElementProperties).
  + **Game Actions:** The [Game ActionSOs](#_Game_Action_1) which will be triggered with the button press, see [Game Action](#_Game_Action).
  + **Action IDs:** IDs of [Game Actions](#_Game_Action_1) which will be triggered with the button press, see [Game Action](#_Game_Action). **NOTE:** Can be used to refer to game actions within other objects, like Dialogs.
  + **Prerequisites:** The prerequisites that determine if the button is clickable.

## Professional Field

Stored in ‘Hamina 1810/Assets/Resources/Content/ProfessionalFields’.

Professional fields work as objects that collect all experience from professions linked to them. E.g., Doctors and nurses might go under the professional field of health care and working as a nurse would accumulate experience to the health care professional field object.

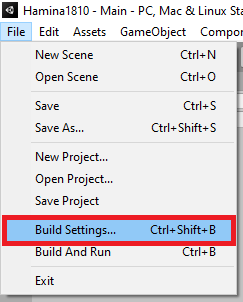
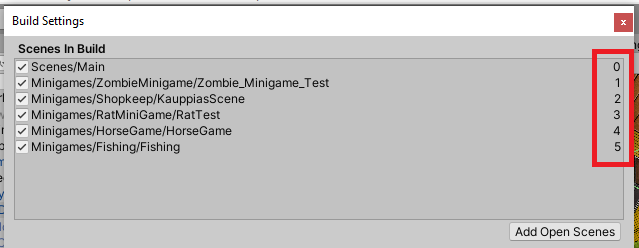
* **Experience:** The initial experience value the player starts with.
* **Name:** Name of the professional field. E.g., Health care, IT, etc.
* **Job Application SO:** The Job Application object used for all professions linked to this field.
* **Weighted Keywords:** Lists of weighted keywords used in all professions linked to this field. The profession’s own weighted keywords override the weights set in the professional field.

## Profession

Stored in ‘Hamina 1810/Assets/Resources/Content/Professions’.

Professions use a hidden resource called Profession Points. These points can be manipulated with the PROFESSION\_CHANGE\_POINTS GameAction. The points are also automatically changed if the player arrives to work late.

Profession objects are used to define jobs with the following properties:

* **Field SO:** The [ProfessionalFieldSO](#_Professional_Field) that this profession falls under.
* **Name:** The name of the profession.
* **Employer Name:** The employer’s name.
* **Employer Image:** Employer Image.
* **Address:** The address of the workplace.
* **Job Description:** A description of the profession.
* **Job Image:** An image representing the job.
* **Required Stats:** Stats required to apply for the job.
* **Required Traits:** The traits required to apply for the job.
* **Required Experience:** The amount of experience required in the **Field SO** ([Professional Field](#_Professional_Field)) to be able to apply for the job.
* **Pay Per Hour:** The amount of money the player gets per hour for their work. Can be overridden by minigames.
* **Experience Per Hour:** The amount of experience per hour of work the Professional Field accumulates.
* **Work Time (Deprecated, use Work Starts and Work Ends instead):** The amount of time that working takes. The game skips forward in time and calculates pay, experience and resources based on this.
* **Work Days:** The weekdays when working is allowed.
* **Work Starts:** The time of day when work starts.
* **Work Ends:** The time of day when work ends.
* **Points Lost Per Hour Late:** Profession points lost per hour late.
* **Point Actions:** An array of conditional [Game Actions](#_Game_Action_1) that are triggered based on the player’s profession points for this profession.
  + **Only Triggered Once:** Determines whether the [Game Action](#_Game_Action_1) is only triggered the first time the profession points go between the Min and Max values.
  + **Resets When Changing Profession:** If true, resets a triggered flag and allow the [Game Action](#_Game_Action_1) to be triggered again if Only Triggered Once is enabled.
  + **Min:** The minimum value of the profession point range.
  + **Max:** The maximum value of the profession point range.
  + **Game Action SO:** The [Game Action](#_Game_Action_1) that will be triggered if the current profession points are between the Min and Max values.
* **Resource Change Per Hour:** The numeric values for all resource changes per hour of work.
* **Minigame Index:** The minigame index number found in build settings. The minigame’s scene must be added in the build settings or the game will be unable to automatically load the minigame. The scenes can be arranged by dragging and dropping but be aware that this changes the index numbers. See pictures below.
* **Application Bonus Chance:** The default bonus given to the chance of getting the positive outcome when sending the job application. 0.25 = 25%, 0.5 = 50%, etc.
* **Required Application Points:** The required application points for a 100% chance of getting the positive application outcome.
* **Max Application Experience:** The maximum amount of professional field experience that is counted towards the application points.
* **Required Interview Score:** The required job interview score to get the positive outcome (Positive Interview Outcome).
* **Positive Interview Action:** The [Game Action](#_Game_Action_1) triggered if the interview outcome is successful.
* **Negative Interview Action:** The [Game Action](#_Game_Action_1) triggered if the interview outcome is unsuccessful.
* **Weighted Keywords:** Lists of highest priority weighted keywords used in this profession.
* **Successful Application Trigger Table:** The [trigger table](#_Random_Event_Trigger) that is triggered on a successful application.
* **Unsuccessful Application Trigger Table:** The [trigger table](#_Random_Event_Trigger) that is triggered on an unsuccessful application.
* **Workday Background:** The background image for the work UI.
* **Workday SO:** The [WorkdaySO](#_Workday) object for this profession.
* **Work Lunch Options:** A list of lunch options for this profession. On top of these, all consumable items in the player’s inventory will automatically be listed as lunch options.

## Quest Category

Stored in ‘Hamina 1810/Assets/Resources/Content/QuestCategory’.

Quest Categories are used to sort and group quests in the quest UI. A Quest Category object contains the following properties:

* **Name:** The name of the category.
* **Background Image:** The background image of the category element in the quest UI.
* **Order Index:** The order number of the category. The categories are listed in order, zero being the first, one being the second and so on.

## Quest

Stored in ‘Hamina 1810/Assets/Resources/Content/Quests’.

Quest objects are used to define quests with the following properties:

* **Name:** Name of the quest. Visible in the quest list.
* **Quest Category SO:** The [Quest Category](#_Quest_Category) this quest will be listed under.
* **Description:** A description of the quest.
* **Prerequisites:** The [prerequisites](#Prerequisites) that must be fulfilled before the quest can be started.
* **Starting Action:** The [Game Action](#_Game_Action) that is triggered when the quest is started. Quests are started with Game Actions: [QUEST\_GIVE](#QUEST_GIVE).
* **Finishing Action:** The [Game Action](#_Game_Action) that is triggered when the quest is finished. Quests are finished with Game Actions: [QUEST\_FINISH](#QUEST_FINISH).

## Random Event

Stored in ‘Hamina 1810/Assets/Resources/Content/RandomEvents’.

Random events are [Game Actions](#_Game_Action) that are picked randomly from the pool of random events based on the trigger type, trigger weight and the global random event trigger chance. Random events have the following properties:

* **Event Type:** Determines whether the event will be considered neutral, bad, or good.
* **Trigger Weight:** The numeric value for the weight (chance to trigger) of the event. E.g., The pool combined weight is 100 and this event’s weight is 30, therefore the chance to trigger is 30 / 100 = 0.3 (30%).
* **Trigger When:** The game or player state when the event can be triggered.
  + **AUTO\_CHECK:** The default state of **Trigger When**, includes **IDLE** and **WALKING** states.
  + **ANYTIME:** The event can trigger at any time.
  + **WALKING:** The event can only trigger when the player is walking.
  + **IDLE:** The event can only trigger when the player is idle.
  + **DIALOG:** The event can only trigger after a dialog is closed.
  + **WORK:** The event can only trigger after working.
  + **SLEEP:** The event can only trigger after sleeping.
* **Prerequisites:** The [prerequisites](#Prerequisites) that must be fulfilled before the random event can be triggered.
* **Game Action SOs:** The [Game Actions](#_Game_Action) that will be triggered when the event is triggered.

## Random Event Trigger Table

Stored in ‘Hamina 1810/Assets/Resources/Content/RandomEventTriggerTables’.

When triggered, an item is randomly selected from the pool based on trigger weights and then triggered.

Random Event Trigger Table contains the following properties:

* **Random Events:** An array of weighted [Random Event Trigger Table Items](#_Random_Event_Trigger_1).
  + **Random Event Trigger Table Item:** A [Random Event Trigger Table Item](#_Random_Event_Trigger_1) object.
  + **Trigger Weight:** The trigger weight for the item above.

## Random Event Trigger Table Item

Stored in ‘Hamina 1810/Assets/Resources/Content/RandomEventTriggerTableItems’.

The item used in [Random Event Trigger Tables](#_Random_Event_Trigger).

The Random Event Trigger Table Item contains the following properties:

* **Game Event:** An array of Game Events that will be activated when this item is triggered.

## Random Work Event Table

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/RandomWorkEventTables’.

A random event table specifically used in the [Workday’s](#_Workday) Challenge Table.

Contains the following properties:

* **Random Events:** An array of weighted [Random Work Event Table Items](#_Random_Work_Event_1). **Note:** To have a chance of skipping the challenge stage, leave an item slot empty with a weight greater than zero.

## Random Work Event Table Item

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/RandomWorkEventTableItems’.

The Items used in [Random Work Event Tables](#_Random_Work_Event). Contains the following properties:

* **Text:** A [dynamic text](#_Dynamic_Text) that will be shown in the work UI when this item is randomly chosen.
* **Image:** The image that will be shown in the work UI when this item is randomly selected.
* **Triggers Always:** A [Game Action](#_Game_Action_1) that will trigger no matter which option the player chooses.
* **Work Challenge Option SO:** An array of [Work Challenge Options](#_Work_Challenge_Option).

## Workday

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/Workdays’.

The root object of a workday. A workday consists of 4 stages: a start stage, a lunch stage, an optional challenge stage and an end stage. Workday objects are used in [Professions](#_Profession). They contain the following properties:

* **Start Stage SO:** The [Start Stage](#_Work_Start_Stage) object for this workday.
* **Challenge Table:** A [Random Work Event Table](#_Random_Work_Event) object that determines how often challenges happen during the workday and what happens when they do.
* **End Stage SO:** The [End Stage](#_Work_End_Stage) object for this workday.

## Work Start Stage

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/WorkStages/Start’.

This object type determines the start stage of a workday that it’s linked to. It contains the following properties:

* **Dynamic Stage Text:** The dynamic text that will be shown in the work UI.
* **Image:** The image that will be shown in the work UI.
* **Stage Options:** An array of [Work Start Options](#_Work_Start_Option) that will be listed in the work UI as buttons for the player to click.

## Work End Stage

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/WorkStages/End’.

This object type determines the end stage of a workday that it’s linked to. It contains the following properties:

* **Dynamic Stage Text:** The dynamic text that will be shown in the work UI.
* **Image:** The image that will be shown in the work UI.

## Work Start Option

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/WorkStageOptions/Start’.

An option for the [start stage](#_Work_Start_Stage) of a [workday](#_Workday). Contains the following properties:

* **Button Text:** The text that will be displayed on the option button.
* **Resource Multiplier:** The resource and pay multiplier that will be applied for the player’s current workday if this option is selected. 0.25 = 25%, 1 = 100%, etc.

## Work Challenge Option

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/WorkStageOptions/Challenge’.

An option for the challenge stage of a [workday](#_Workday). **Warning:** Make sure one of the options doesn’t consume resources, as the challenges are picked at random and cannot properly be taken into account when calculating the required resources for a workday. The work challenge option contains the following properties:

* **Button Text:** The text that will be displayed on the option button.
* **Result Text:** The text that is displayed after the player selects this option.
* **Result Action SO:** The Game Action that will be triggered after the player selects this option.
* **Prerequisites:** The prerequisites that must be fulfilled for the player to be able to select this option. If not fulfilled the button will not be interactable.

## Work Lunch Option

Stored in ‘Hamina 1810/Assets/Resources/Content/Work/WorkStageOptions/Lunch’.

An option for the lunch stage of a [workday](#_Workday). On top of these lunch options, all consumables in the player’s inventory will be listed in the lunch stage UI. The Work Lunch Option contains the following properties:

* **Button Text:** The text displayed in the lunch option button.
* **Cost:** The cost of selecting this option.
* **Resource Change:** The change to the player’s resources after selecting this option.

## Sprite Animation

The sprite animation objects are used with the [Animation Controller](#_Animation_Controller).

Sprite Animation contains the following properties:

* **Frames:** An array of the animation’s Sprites.
* **Start Index:** The index number of the animation start frame. E.g., If the start frame is in Element 3, the **Start Index** would be 3.
* **Animation Length:** The length of the animation in seconds.
* **Loop:** If checked, the animation is looped.
* **Flip X:** If checked, the animations frames are flipped on the X-axis.
* **Flip Y:** If checked, the animations frames are flipped on the Y-axis.
* **Fallback Frame:** The frame to which the animation falls back to when the animation stops. E.g., If a character is walking and the animation is stopped in the middle of the animation, the fallback is used instead of having the character stop with one leg in the air.

## Trait

Stored in ‘Hamina 1810/Assets/Resources/Content/Traits’.

Traits modify the player’s attributes and/or resources. The changes are automatically applied when the trait is acquired and reverted when the trait is lost.

Traits include the following properties:

* **Name:** The name for the trait. Visible on the character sheet.
* **Point Cost:** The cost of the trait at the character creation screen.
* **Available In Character Creation:** If checked, this trait will be selectable in the character creation menu.
* **Skill Type:** Determines the type of the trait. This is mainly used with ADJECTIVE, MAIN\_SKILL and SECONDARY\_SKILL [types](#KeywordTypes) in [job applications](#_Job_Application) and [CVs](#_CV).
  + **NONE:** This type of trait will be considered to be an adjective.
  + **MAIN:** This type of trait will be considered to be a main skill.
  + **SECONDARY:** This type of trait will be considered to be a secondary skill.
* **Required Class:** A social class required by the trait.
* **Excluded Class:** A social class excluded by the trait.
* **Attributes:** Attribute changes when the trait is acquired.
* **Resource Change:** Resource changes when the trait is acquired.
* **Positive:** If checked, the trait is considered to be positive. Affects the trait icon in the character sheet.
* **Application Weight:** The value of this trait, if used in a [job application](#_Job_Application).

## Trigger

Stored in ‘Hamina 1810/Assets/Resources/Content/Triggers’.

Triggers are used to trigger Game Actions in certain situations, determined by the type of the trigger. Only active triggers can trigger. To activate triggers, use the [Activate Triggers](#ActivateTriggers) section of a [Game Action](#_Game_Action_1).

Contains the following properties:

* **Trigger Type:** Determines when triggered.
  + **NONE:** Will not trigger.
  + **DIALOG\_TRIGGER\_COLLISION:** Triggers when the player collides with an object that contains a [Dialog Trigger](#_Dialog_Triggers) and a 2D collider.
  + **FIND\_JOB\_LOCATION:** Triggers when location tracking is started.
  + **SHOP\_COLLISION:** Triggers when a player collides with a game object that contains a shop.
  + **SHOP\_OPEN:** Triggers when a shop is opened.
  + **INVENTORY\_OPEN:** Triggers when the inventory is opened.
  + **PROFESSION\_CHANGE:** Triggers when the player’s profession is changed.
  + **SCENE\_CHANGE:** Triggers right after the scene has changed.
  + **DEATH\_HEALTH:** Triggers when the player dies to lack of health.
  + **DEATH\_HUNGER:** Triggers when the player dies to hunger.
  + **DEAT\_SANITY:** Triggers when the player dies to lack of sanity.
* **Starts Active:** If checked, the trigger starts active and can be triggered immediately.
* **Do Not Deactivate:** If checked, the trigger will not be automatically deactivated after triggered.
* **Game Actions:** The [Game Actions](#_Game_Action_1) that will be executed when this trigger object is triggered.

# Other Useful Classes

## Dynamic Text

Dynamic text is, as the name suggests, a dynamically created text based on either current or saved data. Dynamic text is used, for example in popups and dialogs.

Dynamic Text contains the following properties:

* **Text:** The text that will be displayed in the target text element. The text uses indexed tags that consist of an index number surrounded by curly brackets (e.g., {0}). The tags are automatically replaced by the selected type before the text is displayed, the element 0 replacing {0}, element 1 replacing {1}, and so on. One index can be used multiple times in the text, but **all tags must have a corresponding element in the array**, or the system will fail and cause an exception, breaking the game.
* **Array of Types:** The types that can be used to replace tags in the text.
  + **NONE:** No text.
  + **CURRENCY\_NAME:** The currency name stored in the [GameManager](#_Game_Manager).
  + **PLAYER\_NAME:** The player’s name.
  + **PLAYER\_CLASS:** The player’s class.
  + **PLAYER\_ADDRESS:** The current address of the player.
  + **CURRENT\_TIME:** The current in-game time. Format example: “2 d, 22 h, 22 min, 22 s”.
  + **CURRENT\_TIME\_HH\_MM:** The current in-game time. Format example: “22:22”.
  + **CURRENT\_WEEKDAY:** The current in-game weekday. Format example: “sunnuntai”.
  + **CURRENT\_DATE:** Current in-game date. Format example: “Su 24. toukokuu 2020”.
  + **TIME\_UNTIL\_WORK:** The time until work starts. Format example: “2 d, 22 h, 22 min, 22 s”.
  + **TIME\_LATE:** The amount of time the player is currently late for work. Format example: “2 d, 22 h, 22 min, 22 s”.
  + **CURRENT\_SALARY:** The player’s current salary (for one full workday) as an integer.
  + **TIMES\_STUDIED:** The number of times the player has studied the target [Education](#_Education).
    - **Education SO:** The target [Education](#_Education).
  + **SAVEABLE:** A saveable text type. Custom dynamic text is saved with the [DYNAMIC\_TEXT\_ADD\_CUSTOM](#DYNAMIC_TEXT_ADD_CUSTOM) [Game Action](#_Game_Action_1). If selected, the following properties will become visible:
    - **Text Name:** The name of this custom text. **This has to be unique.** This is the identifier for the previously saved text.
    - **Custom Text Type:** The type of the saveable text.

## Game Time

Game Time is a class used to keep track of different times in the game. Most everything that requires to keep track of time or store a time, uses the GameTime class. GameTime contains the following properties:

* **Days:** Number of days
* **Hours:** Number of hours
* **Minutes:** Number of minutes
* **Seconds:** Number of seconds

## Timer

Timer class itself cannot be used from the editor but is sometime used automatically. For example, if you’re asked to define a time type (absolute or relative) there will most likely be a Timer taking care of the timekeeping.

Generally speaking, the time type defines the absolute target time. If time type is defined as:

* **ABSOLUTE (or TIME\_OF\_DAY):** The given time will be the final target time.
* **RELATIVE (or FROM\_NOW):** The given time will be added to the current in-game time when the timer starts.
* **ABSOLUTE\_TIME\_RELATIVE\_DAY:** The hour, minute and second parts of the given time will be handled as absolute, and the days will be relative. E.g., if the given time is 1 d, 12 h, 0 min, 0 sec: the timer will stop/trigger an action at noon the next day.

## Interaction Mark

The Interaction Mark is an element designed to help guide a player. By default, the interaction marks are automatically generated for Dialog Triggers and Locations based on their current properties. The types of interaction marks are prioritized from highest to lowest priority as follows:

* **QUEST:** If a quest is present, the quest icon will be displayed.
* **SPECIAL LOCATION:** If the location is specified as a special location (Kela or TE-toimisto) an icon unique to them will be displayed.
* **HOME:** If the location is the player’s current home, the home icon will be displayed.
* **SHOP:** If the dialog trigger has a shop, the shop icon will be displayed.
* **CURRENT\_JOB:** If the player’s current job is located here, the current job icon will be displayed.
* **WORKPLACE:** If the location is a workplace, the workplace icon will be displayed.
* **RENTABLE:** If the location is rentable, the rentable icon will be displayed.
* **DIALOG:** If the dialog trigger has active dialog, the dialog icon will be displayed.

Text

Description automatically generatedCurrently, the only way to change the icon references is to edit the InteractionMark.cs script file that is located in the ‘Assets/Scripts/UI’ folder. To change an icon reference, just edit the text between the quotes inside the Initialize method (lines 45-54, image below).

If instead of icons, colors should be used for the interaction mark, uncheck Use Interaction Mark Icons in the [Game Manager](#_Game_Manager).

## Prerequisites

Prerequisites are most often used to determine if a [popup](#_Popup_Container) button or a [dialog](#_Game_Action) button will be clickable.

* **Unemployed:** If checked, the player must be unemployed.
* **Required Profession SO:** The player must be working in this [profession](#_Profession).
* **Required Profession Text Override:** If the Required Profession requirement is not met, this override text will be used instead of the default, unless left empty.
* **Prohibited Profession SO:** The player must not be working in this [profession](#_Profession).
* **Prohibited Profession Text Override:** If the Prohibited Profession requirement is not met, this override text will be used instead of the default, unless left empty.
* **Quest Started:** The player must have started these [quests](#_Quest).
* **Quest Completed:** The player must have completed this [quests](#_Quest).
* **Item Acquired:** The items that need to be in the player’s inventory.
* **Money Owned:** The amount of money the player needs to have.
* **Stats:** Required stats.
  + **Stat:** The required stat (strength, intelligence, etc.)
  + **Equality:** How to compare the players stats with the given value (lesser than, equal or greater than).
  + **Value:** The numeric value the player’s stat is compared to. (e.g., Is strength greater than 3?)
* **Required Resources:**
  + **Resource Type:** The required resource (health, stamina, etc.)
  + **Equality:** How to compare the player resources with the given values (lesser than, equal or greater than).
  + **Value:** The numeric value the player’s resource is compared to. (e.g., Is health greater than 50?)
* **Required Traits:**
  + **Trait Name:** The name of the required trait. Needs to be EXACT. Can be left empty if a trait scriptable object is used instead (below).
  + **Trait:** A scriptable object of the required trait.
  + **Is Active:** Selected if the trait must be active, not selected if the trait must be inactive.
* **Required Educations:**
  + **Education SO:** The required [education](#_Education) scriptable object.
  + **Started:** If checked, the education must be started. If not checked, the education must not be started.
* **Required Class:** Required player class.
* **Excluded Class:** Excluded player class.