Sommario

[Gdevelop Point and click system 0.2 ITA 2](#_Toc50739196)

[Let’s start 2](#_Toc50739197)

[GDPacs: il giusto approccio 2](#_Toc50739198)

[Modifica Intro 3](#_Toc50739199)

[Aggiunta lingua: 3](#_Toc50739200)

[Aggiunta slot di salvataggio 3](#_Toc50739201)

[Scene L1S1 4](#_Toc50739202)

[Elementi 4](#_Toc50739203)

[Oggetti solo di riempimento grafico 4](#_Toc50739204)

[Elementi GDPACS nelle scene di gioco 5](#_Toc50739205)

[Item custom 7](#_Toc50739206)

[Dialoghi: 8](#_Toc50739207)

[Qualche esempio utile: 10](#_Toc50739208)

[Variabili globali che ha senso modificare 15](#_Toc50739209)

[Gdevelop Point and click system 0.2 ENG 16](#_Toc50739210)

[Let’s start 16](#_Toc50739211)

[Analyze the objects, their associated **variables** and the groups to which they belong. 16](#_Toc50739212)

[GDPacs:  a very good start 16](#_Toc50739213)

[Modify Intro Scene 17](#_Toc50739214)

[Add a language: 17](#_Toc50739215)

[Added save slot 17](#_Toc50739216)

[Scene L1S1 18](#_Toc50739217)

[ElementS 18](#_Toc50739218)

[Elementi GDPACS nelle scene di gioco 18](#_Toc50739219)

[Item custom 21](#_Toc50739220)

[Dialogues: 21](#_Toc50739221)

[FAQ: 23](#_Toc50739222)

[DEMO Source 29](#_Toc50739223)

# Gdevelop Point and click system 0.2 ITA

Il progetto ha lo scopo di minimizzare il coding in fase implementativa per i giochi Point and click sviluppati in GDevelop.

I flussi (semi)automatici gestiscono:

* dialogo (Yarn tool)
* raccolta oggetti
* gestione inventario
* gestione lingue
* gestione cambio di scene
* gestione suoni (ON/OFF)
* salvataggio status ai checkpoint (con differenti salvataggi )
* oggetti che si spostano in parallasse ( distanti e vicini )
* oggetti mostrati davanti o dietro il player
* animazioni diverse in base al travestimento del Player
* limiti di inquadratura della camera sull’asse x

## Let’s start

Un consiglio: giocate una partita alla demo. Finito il gioco, rifatela pensando agli elementi e le azioni che avete incontrato: dialoghi, raccolta oggetti, uso oggetti, cambio di scena, movimenti forzati.

Riflettete

Bene ora, copiate il progetto in uno nuovo dando un nome più simpatico.

Aprite il codice analizzando solo le parti CUSTOM nella scene L1S1:





….

Analizzate gli oggetti presenti nelle 3 scene, le loro **variabili** associate e i gruppi ai quali appartengono.

## GDPacs: il giusto approccio

Il primo step da compiere è quello di copiare il progetto GDpacs, e giocarci un minimo per capire come è il gioco. Dopo aver analizzato gli elementi il passo due è personalizzare le scene INRO e L1S1 in base alle proprie esigenze grafiche.

Mantenendo la logica del gioco, sostituite le animazioni degli sprite. Solo quelle.

Modificarne SOLO GRAFICAMENTE le due principali scene:

* Intro: schermata dove scegliere lingua e il file di salvataggio dal quale ripartire
* L1S1: scena di gioco.

Quando il gioco è personalizzato con la vostra grafica provarlo sempre con le stesse logiche del gioco.

Se tutto ok poi passerete a costruire il vostro mondo.

Il gioco sarà composto da diverse scene: l’intro e le varie “stanze”; l’intro è presente come scene INTRO; la prima scene di gioco è la L1S1. (nome definito nella variabile **LevelSceneNumber**). Il punto dove il protagonista inizierà la partita sarà dove avete posto l’oggetto StartPosition, animazione sp1 

Inizialmente suggerisco di modificare solo gli elementi già presenti dal punto di vista della grafica lasciando un personaggio “Receptionist” e i pochi oggetti presenti.

Solo quando si è soddisfatti dal punto di vista estetico, modificare la storia della prima scena in base alle proprie esigenze.

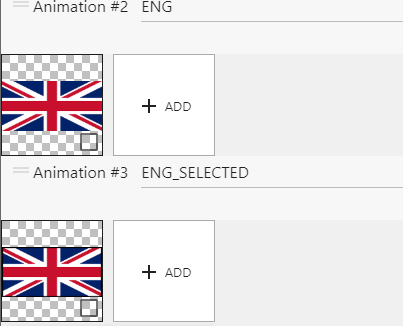
## Modifica Intro

Modificare lo sfondo e le icone delle bandiere.

Aggiunta lingua: Nel caso in cui due lingue non fossero sufficienti, aggiungere le relative animazioni. Per ogni lingua sono previste 2 animazioni.

Nome\_lingua (es ITA)

nome\_lingua\_selected (es ITA\_SELECTED).



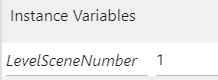
Se si ha necessità di aggiungere nuove lingue, includere le **coppie** di animazioni con la stessa logica e creare nuove istanze. Il file Yarn dei dialoghi dovrà avere il nome dell’animazione (ad esempio per l’Italia è ITA)- Lo vedremo nei prossimi capitoli.

Aggiunta slot di salvataggio Ad inizio partita è possibile scegliere se iniziare una nuova partita o proseguire con una precedente. Il numero dei salvataggi disponibili è settato a 3.

Per aggiungerne, trascinare un nuovo Match nella schermata e un nuovo NewMatchText. Quest’ultimo ha una variabile dal nome LevelSceneNumber. Essa indica il valore dello slot selezionato.

Gli slot ora presenti sono popolati con “1” per il primo “2” per il secondo e “3” per il terzo. Indovina come chiamare il quarto ?



Variabile del testo: 

*Se attivato il* ***DebugMode****, utilizzare il tasto giallo a run-timr per cancellare le precedenti partite salvate; è utile nei test. Per attivare il* ***DebugMode****, valorizzare la viariabile globale così:*

**

## Scene L1S1

L1S1 è la scena principale di gioco (leve 1 Scene 1).

Tutte le scene di gioco avranno lo stesso codice (ad eccezione delle parti custom). Suggerisco di realizzare bene la prima scene in base alle esigenze grafiche per poi copiarla e usarla come template per ogni nuova scene.

Elementi principali presenti in ogni scene di gioco:

Elementi **DebugMode** : elementi visibili e utilizzabili solo in debug per testare meglio il gioco. Il gioco compilato non li mostra. In fase di preview può essere utile visualizzarli impostando la variabile globale DebugMode = “Yes”. Gli elementi visibili in modalità debug sono:

* ClearMemory (**DebugMode**): è un tasto giallo che serve per svuotare lo stato del gioco al fine di partire sempre come se la partita fosse appena iniziata; se il Player va sul checkPoint viene memorizzato in automatico lo stato di ogni elemento della scene; può essere utile un reset di tali informazioni.
* Debug (**DebugMode**): scritte che aiutano a capire cosa sta succedendo.

### Oggetti solo di riempimento grafico

E’ tutto quanto non ha interazione con il giocatore e ha il solo obiettivo di arricchire il gioco di elementi grafici. Ad esempio nella Demo troviamo il tavolo dove è posata la parrucca. In fase di movimento del player essi si possono spostare:

* più lentamente del giocatore: ad esempio le montagne di sfondo. Creare un oggetto e abbinarlo al gruppo “BackGround”
* come il giocatore ma stando davanti ad esso: Creare un oggetto e abbinarlo al gruppo “BehindOfThePlayer”
* come il giocatore ma stando dietro ad esso: Creare un oggetto e abbinarlo al gruppo “InFrontOfThePlayer”
* più velocemente del giocatore: ad esempio una colonna. Creare un oggetto e abbinarlo al gruppo “Foreground”

Per associarvi gli oggetti aggiungerli ai gruppi: attenzione per gli oggetti che hanno ruoli attivi nel gioco associarli solo a “BehindOfThePlayer” o “InFrontOfThePlayer”

Un altro gruppo che identifica la visibilità degli Item è “HideOnStart”. Se un oggetto è associato ad esso, alla partenza sarà invisibile e sarà necessario mostrarlo solo a run-time (vedi oggetto Obj\_Smile)

Elementi GDPACS nelle scene di gioco: elenco degli elementi utilizzato da GDPACS. Sono fondamentali per gestire la partita.

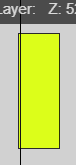
Attenzione: Anche se non usati, non rimuoverli dell’elenco degli oggetti disponibili.

Gli elementi presenti nella demo sono:

* **Basement**: pavimento con behavior; deve essere presente nella scene. Modificare solo l’animazione
* **Basement2**: pavimento senza behavior; modificare solo l’animazione (o rimuovere l’istanza se nonnecessario)
* **MovePlayerLeft & MovePlayerRight & MoveAction**: tasti per GUI; se premuto forza il tasto left; modificare solo l’animazione. Se non serve nasconderlo.
* **SaveCheckPoint**: icona per salvataggio stato partita; se il giocatore lo collide vengono salvate le informazioni del gioco. Non è visibile pertanto suggerisco di non modificarla

 a meno che non la si voglia renderla visibile

* **PlayerBox**: è il Player vero e proprio; è l’oggetto che si muove e che collide. Movimenti e Hitbox sono gestiti da qui. Consiglio di modificarne solo dimensioni e hitbox per le proprie esigenze

 Non è visibile pertanto suggerisco di non modificarla

* **Player**: è la cover del Player, solo la grafica; modificare solo le animazioni. Attenzione, i nomi delle animazioni sono composti da 2 parole: il “travestimento” del player e l’azione.

Ad esempio **Normal**Idle è l’animazione del vestito normale quando il giocatore è fermo

**Punk**Walk è l’animazione del giocatore con sembianze Punk mentre cammina

Nel caso di esempio esistono due tipi di travestimenti: “normal” e “punk”. Per ognuno di essi esistono diversi movimenti.

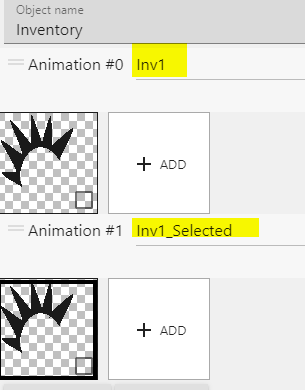
Per aggiungere travestimenti mantenere la stessa logica.

Nel codice, quando si ha la necessità di “travestire” il giocatore in modo diverso, forzare la variabile di oggetto del Player Setting- AnimationSet con il nome del travestimento.

Ad esempio:



* **Inventory**: inventario mostrato a video; modificare solo le animazioni. Per ogni nuovo oggetto aggiungere l’animazione Inv1 (1 è un contatore quindi il primo è Inv1, il secondo Inv2..) e Inv1\_Selected per definire un oggetto quando è selezionato. Il legame tra l’oggetto e l’animazione in inventario è definita in una variabile dell’oggetto stesso. Ad esempio PunkHead ha la variabile “NameOfInventoryAnimation” = Inv1 ( vedremo i dettagli nel prossimo capitolo)

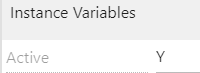


* **MakeAction**: oltre a personaggi e oggetti, è possibile forzare dei punti che possono far scattare un’azione per il solo fatto che il Player li tocchi (ad esempio cammino su un tappeto e finisco in un buco); non sono visibili; l’identificazione dell’azione si basa sul nome dell’animazione.

Ad esempio, aggiungo alla scene questo oggetto, animazione 3

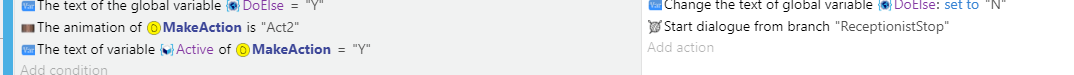


E lo stato attivo



Se inserito, il player passandoci sopra fa scattare una chiamata al codice. E’ necessario customizzarla. Ad esempio:





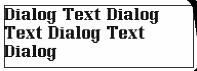
….

aggiungere eventuali altre azioni in animation se 5 non dovessero bastare

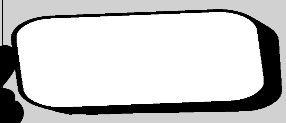
* **StartPosition**: sono i punti da dove il Player potrebbe essere appena entrati nella scene; ad inizio partita è sempre sp1 ma se ad esempio una scene dovesse essere un corridoio ogni porta corrisponderebbe ad uno startPosition. Nel momento in cui il player passa da una stanza al corridoio, lo startPosition della variante verrebbe forzato e pertanto il personaggio si troverebbe sempre davanti alla giusta porta. Anche qui il riconoscimento dello startPosition si basa sull’animazione. Per aggiungere eventuali altri starting point usare la stessa logica di quelli già presenti.

Ogni scene deve avere almeno la sp1 

* **DialogText**: elemento di testo dove mostrare il dialogo. Modificare solo font e dimensioni (Per i testi vedi il paragrafo dialoghi)



* **DialogBox**: cornice del dialogo. Modificare solo l’animazione;



* **DialogIcon**: Icona del personaggio che parla. Modificare e aggiungere le animazioni;



* **MagicWall**: muro invisibile che impedisce di proseguire (per disattivare vedi paragrafo Item custom)

 Da codice Custom è possibile disabilitarlo

* **LimitCameraLeft e LimitCameraRight**: punti sull’asse x che determinano i confini di visualizzazione della telecamera



Item custom: tutti gli altri elementi (Receptionist, Obj\_Curtain, Obj\_PunkHeda) sono sprite che si interfacciano con il giocatore. Ad esempio persone con le quali dialogare, oggetti da prendere e usare.

Per creare un oggetto/personaggio è necessario seguire i seguenti step:

**Personaggio**: copiare tutto l’oggetto da “Reception”. Modificare le animazioni e le variabili di oggetto con questa logica

* KillOnTouch: Y/N con è “Y” e tocchi il personaggio, muori
* DialogueNode: Nome del nodo del dialogo (vedi paragrafo Dialoghi)
* PickUpAble: Posso raccogliere il personaggio? Di default indicare il valore “N”, Non ha senso raccogliere un personaggio
* GoToScene: nel caso di cancelli/porte indica dove trasportare il Player. Nel caso di personaggi, suggerisco di usare il parametro “none”. Non ha senso far saltare il Player in un’altra scene GoToStartingPoint: nel caso di cancelli/porte indica dove trasportare il Player. Nel caso di personaggi, suggerisco di usare il parametro “none”. Non ha senso far saltare il Player in un’altra scene
* NameOfInventory: Quale animazione mostro nell’inventario se raccolgo il personaggio ? Di default indicare il valore “none” perché non ha senso raccogliere un personaggio

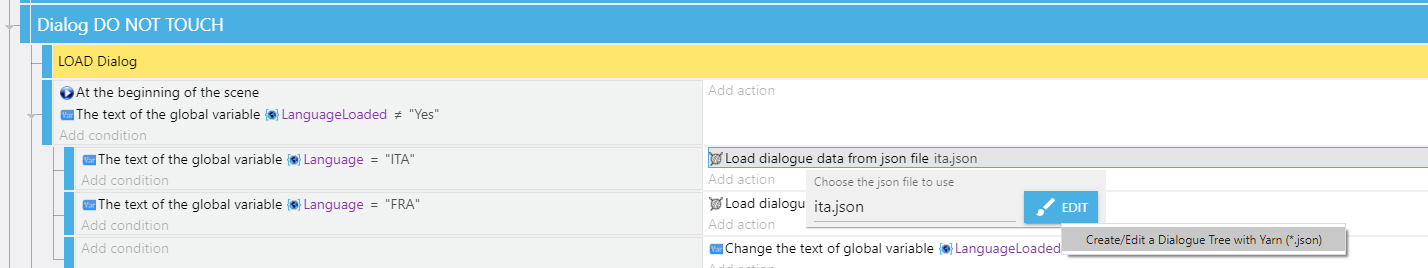
**Oggetto**: copiare tutto l’oggetto da “Obj\_PunkHead”. Modificare le animazioni e le variabili di oggetto con questa logica

* KillOnTouch: Y/N se è Y e lo tocchi, muori
* DialogueNode: Nodo del dialogo (vedi paragrafo Dialoghi). Scatta anche in caso di raccolta dell’oggetto (ad esempio quando raccogli il parrucchino punk)
* PickUpAble: Y/N Se “Y”, in caso di azione posso raccoglierlo.
* GoToScene: in caso di azione, cambia la scena con quella indicata dal valore del parametro (ad esempio un cartello ). Ad esempio il cartello delle S1L1 ha parametro “L1S2”
* GoToStartingPoint: in caso di azione, cambiala scena con quella indicata dal valore del parametro GoToScene e pone il player nel punto “StartPosition” che ha l’animazione che si chiama come il valore indicato in questo parametro(ad esempio un cartello ). Ad esempio il cartello delle S1L1 ha parametro “sp1”
* NameOfInventory: nel caso di possa raccogliere, indica il nome dell’animazione dell’inventario che verrà mostrata. Ad esempio la parrucca punk ha come valore Inv1. Attenzione per limiti tecnici i nuovo elementi in inventario dovranno avere sempre come nome Inv.. + un numero Inv1, Inv2, Inv3….
* OtherAction = nel caso di azioni non previste (vedi esempio dopo) è l’identificativo che serve per capire quale azione compiere

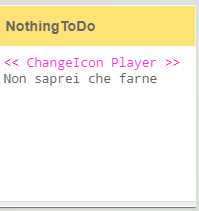
Dialoghi:

I dialoghi sono tutti caricati in un file ***Yarn*** in fase di inizio della scena. In base alla lingua scelta viene caricato un file differente.

Il caricamento avviane in questo punto. Da qui è possibile modificare il file. Attenzione il nome del file deve essere quello indicato nella scene INTRO.



Premendo EDIT e modificare con questa logica:

Ad esempio: 

NothingToDo è il nome del nodo. Deve essere indicato quando si desidera avviare un dialogo con tale testo

 è il comando che fa apparire la faccia del Player a fianco del testo. Tale icona deve essere presente nelle animazioni dell’oggetto **DialogIcon**:

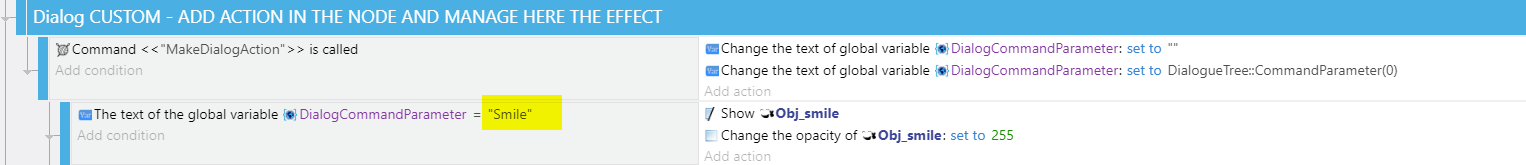
Ad esempio, questo comando nel custom

per far apparire gli oggetti del dialogo, la relativa icona e il testo “Non saprei che farne”

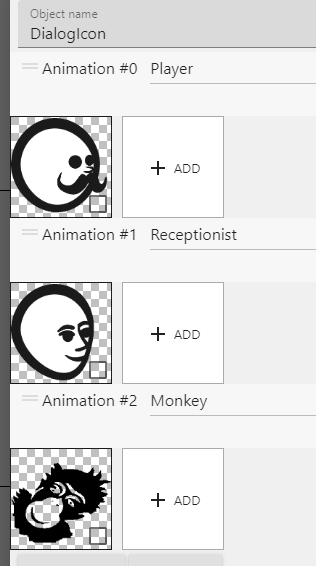
Nei nodi di dialogo è possibile far scattare altre azioni gestite tra i comandi. Ad esempio:



MakeActionSmiel è gestito con il codice:



Basta aggiungere un nuovo blocco e un comando nel nodo per gestire questo tipo di necessità.



## Qualche esempio utile:

#### voglio inserire un oggetto parrucchino che si possa raccogliere

NON è necessario toccare il codice.

Vedi **PunkHead** ( nella Demo ). Per un nuovo oggetto, aggiungerlo al gruppo Item e al BehindOfThePlayer (per vederlo dietro)

Modificare l’animazione cambiandone la grafica

Forzo i parametri nel seguente modo

* KillOnTouch: N (non devi morire se lo tocchi)
* DialogueNode: node (non parla)
* PickUpAble: Y (si può raccogliere)
* GoToScene: none (se lo pigli non passi ad altra scena)
* GoToStartingPoint: none (se lo pigli non passi ad altra scena)
* NameOfInventory: Inv1 (E’ l’animazione in inventario)
* OtherAction = none (non esegue comandi custom)

Modifico l’oggetto **Inventory** aggiungendo due animazioni: **Inv1** che rappresenta l’oggetto e **Inv1Selected** che rappresenta l’oggetto selezionato.

#### creare un oggetto che si possa raccogliere e che mostri una scritta quando raccolto

NON è necessario toccare il codice.

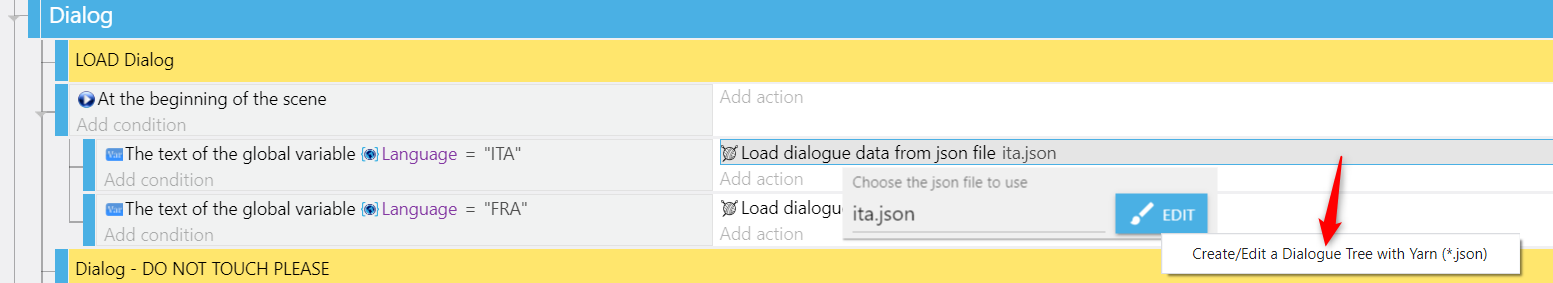
Copia PunkHead (della demo) e lo aggiungo al gruppo Item e al BehindOfThePlayer (per vederlo dietro). Ne modifico l’animazione.

Forzare i parametri nel seguente modo

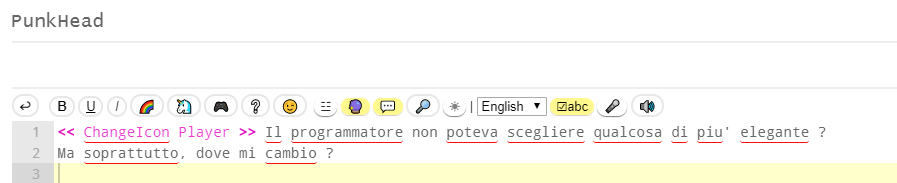
* KillOnTouch: N
* DialogueNode: PunkHead (quando lo raccolgo richiama il nodo PunkHead)
* PickUpAble: Y
* GoToScene: none
* GoToStartingPoint: none
* NameOfInventory: Inv4
* OtherAction = none

Modificare l’oggetto Inventory aggiungendo due animazioni: **Inv1** che rappresenta l’oggetto e **Inv1Selected** che rappresenta l’oggetto selezionato.

Apro il file di dialogo e aggiungo un nodo dal nome “PunkHead”. Esso deve iniziare con l’indicazione dell’icona del Player e in testo. Ad esempio:



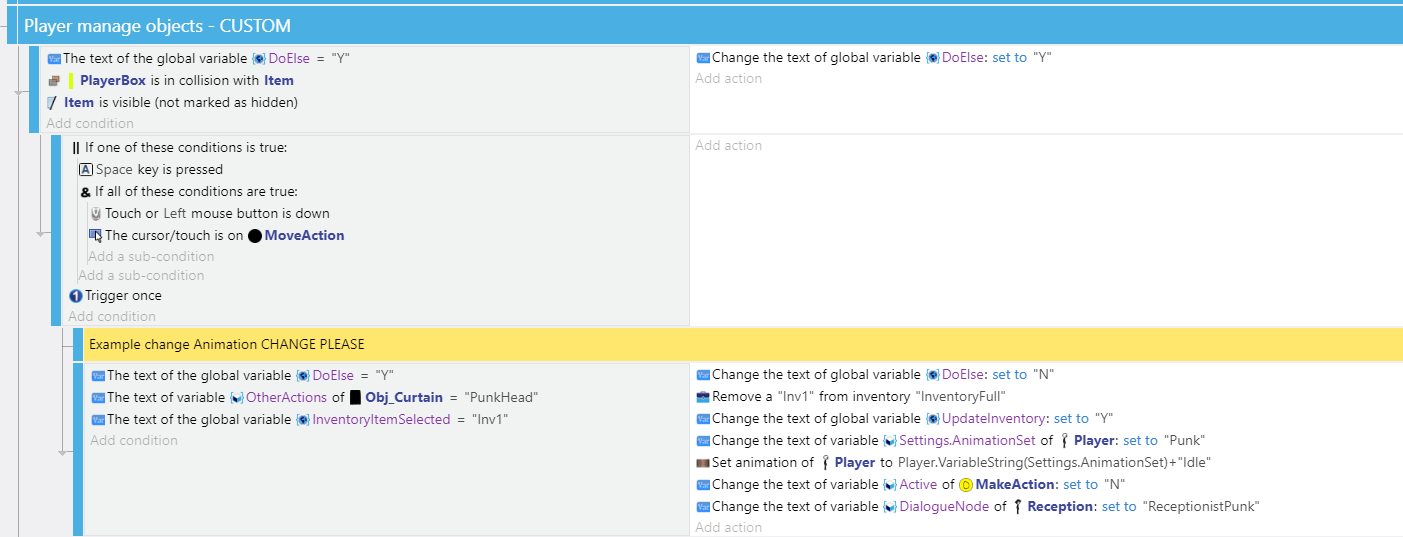
E aggiungere un nodo dal nome “Scudo”. Ad esempio



*<<ChangeIcon Player>> significa che quando appare la scritta verrà mostrata l’icona del giocatore a fianco*

#### Utilizzare un oggetto in inventario con uno non raccolto per modificare l’animazione del Player (travestimento)

** Cerca Example2 nel codice**



….

Modificare solo

L’oggetto selezionato in inventario (InventoryItemSelected)

L’oggetto da usare nella scene

Il nome dell’animazione da togliere dall’inventario (oggetto usato)

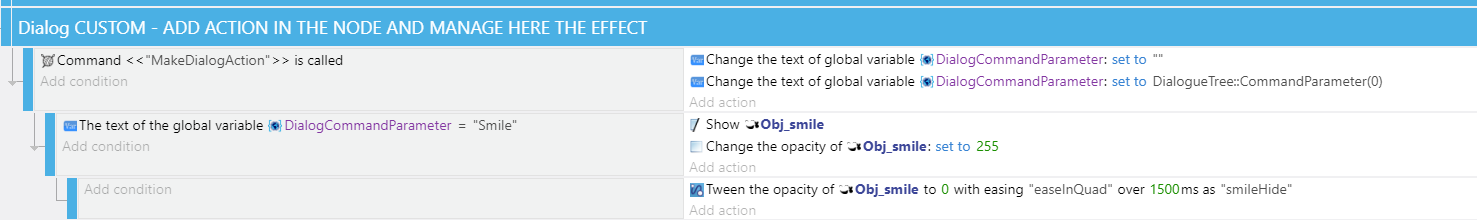
Il nome del travestimento (Punk) che definirà come impostare l’animazione

##### voglio che scatti un dialogo se passo in un determinato punto e mostri uno sprite

** Cerca Example1 nel codice**

Aggiungere un tasto Action, con un’animazione (ad esempio Act2) non ancora usata nella scene e aggiungo il seguente codice custom





Modificare solo quanto in giallo:

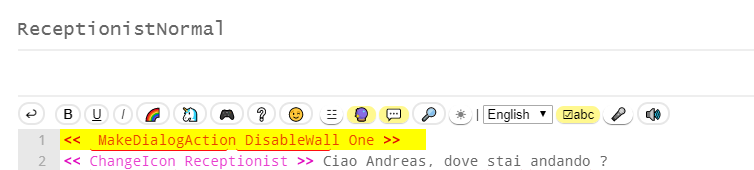
* nome dell’animazione di MakeAction (Act2)
* nome del nodo del dialogo (ReceptionistShop)

disattivazione MagicWall in fase di dialogo. Inserisco il comando nel nodo di dialogo

** Cerca Example3 nel codice**

Inserire un MagicWall e modificare il nome della viariabile **Name** con “One”

Inserire il seguente nodo nel dialogo e assegnarlo ad un Item



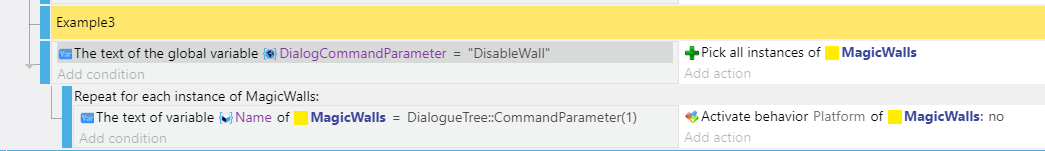
*Makeaction è il comando generico per far accadere un evento*

*DisableWall rappresenta l’evento di disattivazione*

*One è il nome del MagicWall disabilitato*

e aggiungere a codice SENZA modifiche





##### Voglio aggiungere una colonna vicina

Creo un nuovo oggetto (come le Mountains della demo)

Lo aggiungo al gruppo Foreground

Lo inserisco n volte della scena di gioco

#### Aggiungere montagne lontane

Creo un nuovo oggetto (come le Mountains della demo)

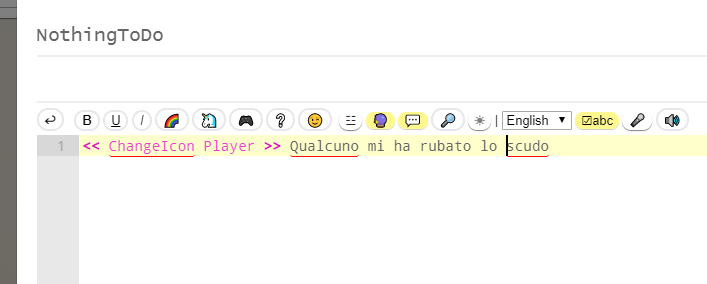
Lo aggiungo al gruppo Background

Lo inserisco n volte della scena di gioco

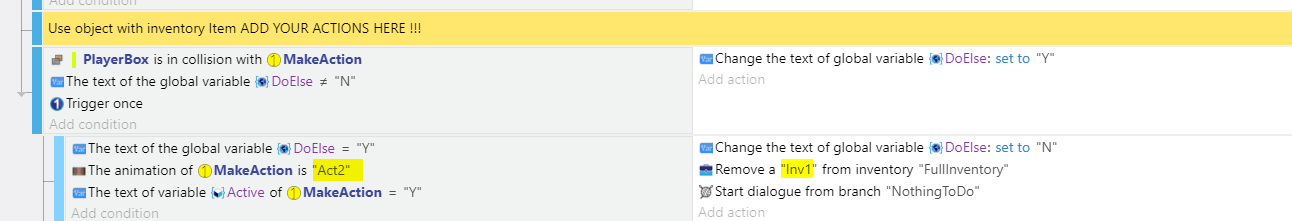
##### Togliere un elemento dall’inventario e forzare un messaggio passando da un punto di azione

Creare un nuovo punto di azione associando un’animazione sin d’ora non usata (es.Act2)

Creare un nuovo Nodo nel dialogo. Ad esempio



Inserire in questo punto il seguente codice:



Le parti in giallo sono da modificare:

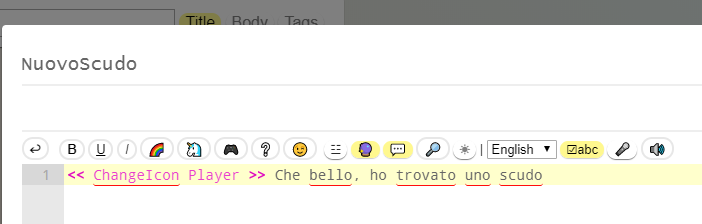
Act2 identifica l’ActionPoint

Inv1 identifica l’oggetto da togliere dall’inventario (Animazione diInventory)

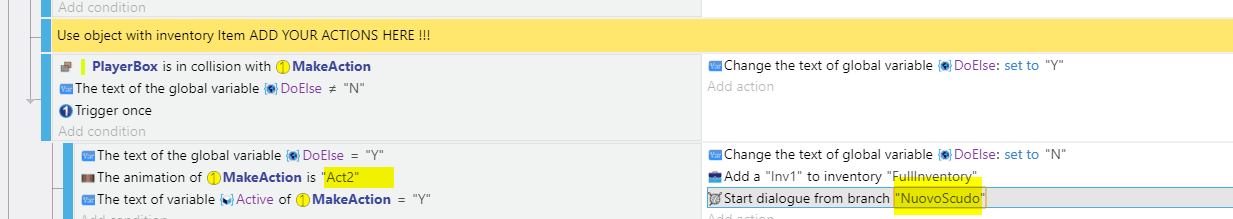
##### Aggiungere un elemento dall’inventario e forzare un messaggio passando da un punto di azione

Creare un nuovo punto di azione associando un’animazione sin d’ora non usata (es.Act2)

Creare un nuovo Nodo nel dialogo. Ad esempio



Inserire in questo punto il seguente codice:



Le unicheparti da cambiare sono in giallo: l’animazione che identifica l’ActionPoint e il nodo del dialogo

## Variabili globali che ha senso modificare

BackgroundSpeed velocità parallasse background

ForegroundSpeed velocità parallasse primissimo piano

MoveParallax abilita parallasse

textScrollSpd: velocità scroll dialogo

DebugMode attiva oggettio di degug

# Gdevelop Point and click system 0.2 ENG

SORRY FOR MY SPAGHETTI ENGLISH

The project aims to minimize the implementation coding during Point and click GD games developed.

The authomatic flows are

* dialog (Yarn tool)
* pick up objects
* manage&use inventory
* languages management
* change of scene
* sound management (ON/OFF)
* checkpoints ( with different saves)
* parallax movements ( near and far )
* objects shown in front or behind the player
* Player animations
* Camera x movements range

## Let’s start

A tip: play a game of the demo. Play again, thinki about the elements and actions you encountered: dialogues, collection of objectuse of objects, change of scene, forced movements. 

Well now, copy the project into a new one and give it a nicer name.

Open the code by analyzing only the CUSTOM parts in the L1S1 scene:





## Analyze the objects, their associated **variables** and the groups to which they belong.

## GDPacs:  a very good start

The first step: copy the GDpacs project and play it to understand how the game is. After analyzing the elements, customize the INTRO and L1S1 scenes according to your graphic needs.

Keeping the logic of the game, replace the sprite animations. Only those.

Modify the two main scenes GRAPHICALLY:

* Intro: screen where you can choose the language and the save file from which to restart
* L1S1: game play

When the game is customized with your graphics, try it using the with the same game play as the demo.

If everything is ok, you will start to building your world.

The game will consist of several scenes: the intro and the various "rooms"; the intro is present as INTRO scenes; the first scene is L1S1. (name defined in the variable **LevelSceneNumber**). The starting point is where there is the StartPosition, animation sp1 

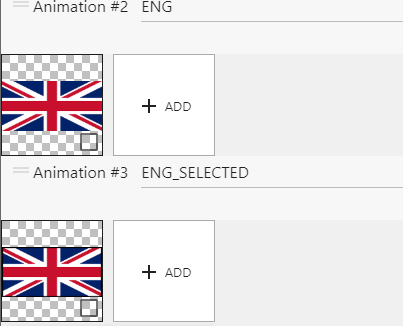
## Modify Intro Scene

Change the background and flag icons.

Add a language: If two languages are not enough, add the corresponding animations. For each language there are 2 animations.

Name\_of\_language (es ITA)

Name\_of\_language\_selected (es ITA\_SELECTED).

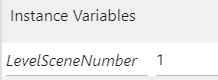


If you need to add new languages, include **pairs** of animations with the same logic and create new instances. The Yarn file must have the name of the animation (for example for Italy it is ITA) - We will see it in the next chapters.

Added save slot At the beginning of the game it is possible to choose whether to start a new game or continue with a previous one. The number of available saves is set to 3.

To add it, let’s drag a new Match to the screen and a new NewMatchText. The latter has a variable named LevelSceneNumber. It indicates the value of the selected slot.

The slots now present are populated with "1" for the first, "2" for the second and "3" for the third. Guess how to call the fourth? 

Text Variables: 

*If* ***DebugMode*** *is activated, use the yellow run-time button to delete the previous saved games; it is useful in tests. To activate the* ***DebugMode****, set the global viable value as follows:*

**

## Scene L1S1

L1S1 is the main game scene (level 1 Scene 1).

All game scenes will have the same code (except custom parts). I suggest to make the first scene using the graphic you need and then copy it and use it as a template for each new scene.

Main elements present in each game scene:

ElementS **DebugMode** : elements visible and usable only in debug to better test the game. The compiled game does not show them. During the preview phase it may be useful to view them by setting the global variable DebugMode = "Yes". The elements visible in debug mode are:

* ClearMemory (**DebugMode**): it is a yellow button that serves to clear the game state in order to always start as if the game had just started; if the Player goes to the checkPoint, the status of each element of the scene is automatically stored; a reset of this information may be useful.
* Debug (**DebugMode**): writings that help to understand what is happening.

Graphic fill-only objects

It is all that has no interaction with the player and has the sole objective of enriching the game with graphic elements. For example, in the Demo we find the table where the wig is placed. During the movement of the player they can be moved:

* slower than the player: for example the mountains in the background. Create an object and match it to the group “BackGround”
* at the same speed as the player but standing in front of it: Create an object and match it to the group “BehindOfThePlayer”
* at the same speed as the player but standing in behind of it: Create an object and match it to the group “InFrontOfThePlayer”
* faster than the player: for example a column. Create an object and match it to the group “Foreground”

To associate objects with them, add them to groups: watch out for objects that have active roles in the game, associate them only with “BehindOfThePlayer” o “InFrontOfThePlayer”

Another group that identifies the visibility of the Items is "HideOnStart". If an object is associated with it, at the start it will be invisible and it will be necessary to show it only at run-time (see object Obj\_Smile)

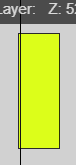
Elementi GDPACS nelle scene di gioco: list of items used by GDPACS. They are essential to manage the game.

Warning: do not remove them from object list.

The elements present in the demo are:

* **Basement**: floor with behavior; must be present in the scene. Change only the animation
* **Basement2**: floor without behavior; modify only the animation (or remove the instance if not necessary)
* **MovePlayerLeft & MovePlayerRight & MoveAction**: GUI keys; if pressed, they force the left/right/up keyd; please modify the animation. If you don't need them, hide them.
* **SaveCheckPoint**: game status save icon; if the player collides with it, the game informations are saved. It is not visible so I suggest not to modify it



* **PlayerBox**: it is the real Player; it is the object that moves and collides. Movements and Hitboxes are managed from here. I recommend to change only its size 
* **Player**: it is the Player cover; please edit the animations. Attention, the animation names consist of 2 words: the player's "dress" and the action.

Example: **Normal**Idle it is the stopped normal suit Player animation **Punk**Walk it is the walnk Punk dress player animation

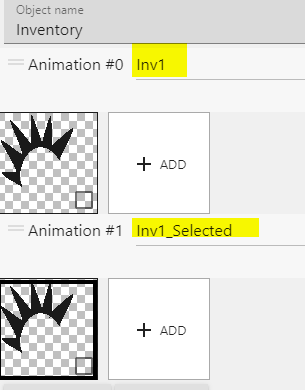
In the example case there are two types of dresses: "normal" and "punk". For each of them there are different movements.

To add dress keep the same logic.

In the code, when you need to change dress player, modify the object variable of the Player Setting- AnimationSet with the name of the disguise.

For example:

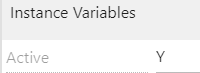
* **Inventory**: inventory shown on screen; edit only the animations. For each new object, add the **Inv1** animation (1 is a counter so the first is Inv1, the second Inv2 ..) and **Inv1\_Selected** to define an object when it is selected. The link between object and the animation in the inventory is defined in a object variable.

Example: PunkHead has the variable "NameOfInventoryAnimation" = Inv1 (we will see the details in the next chapter)

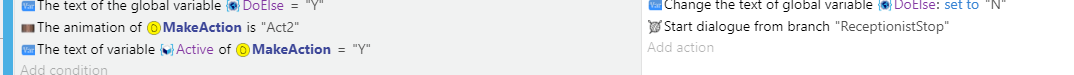
* **MakeAction**: in addition to characters and objects, it is possible to force points that can trigger an action simply because the Player touches them (for example, I walk on a carpet and end up in a hole); they are not visible; the identification of the action is based on the name of the animation.

For example, I add this object, animation 3 to the scene

The state should be active



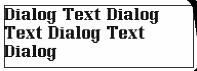
If the player collide with it, it triggers a call to the code. It is necessary to customize it. For example:



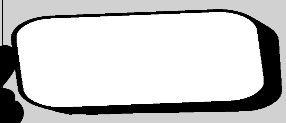
….

If it isnecessary, add other actions in animations.

* **StartPosition**: the Player starts the game on this position. at the beginning of the game it is always sp1 but if a scene ( ex. with a corridor) can start from different points. In this case use the others animations. When the player go from a room to the corridor, the startPosition of the variant would be forced and therefore the character would always be in front of the right door. Here, too, the recognition of the startPosition is based on animation. To add any other starting points, use the same logic as those already present. Each scene must have at least sp1 
* **DialogText**: It is the text element to display the dialogue. Change font and size only (For texts see the dialogue paragraph)



* **DialogBox**: dialogue frame. Please change only the animation;



* **DialogIcon**: Speaking character icon. Please edit and add animations;



* **MagicWall**: invisible wall. It prevents you from continuing (to deactivate, see the Custom Item paragraph)



* **LimitCameraLeft e LimitCameraRight**: they represents the x axis delimitations that determine the camera view.



Item custom: all the other elements (Receptionist, Obj\_Curtain, Obj\_PunkHeda) are the sprites that have a dialog or action with the player. For example, people to talk to, objects to take and use.

To create an object / character, please follow these steps:

**Character**: copy the whole object from "Reception". Edit animations and object variables in this way:

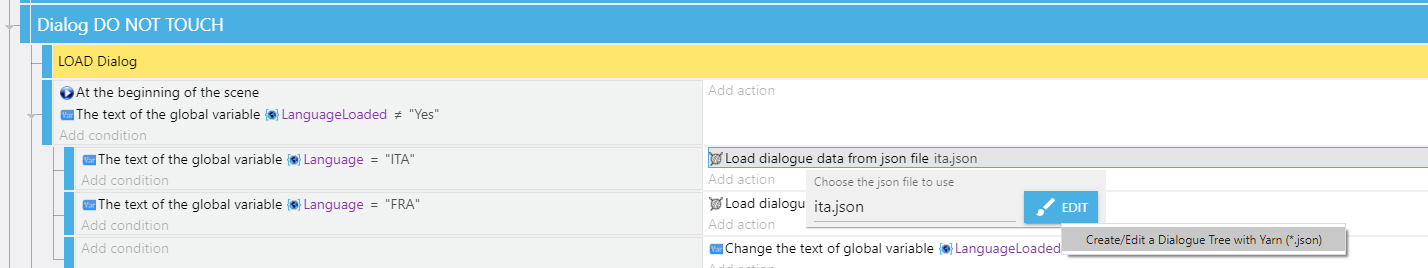
* KillOnTouch: Y/N . “Y” to kill on collide
* DialogueNode: Yarn node name (see next paragraph Dialogues)
* PickUpAble: Can I collect the character? By default indicate the value "N", It makes no sense to collect a character
* GoToScene: in the case of gates / doors it indicates where to transport the Player. In the case of characters, I suggest using the "none" parameter. It makes no sense to make the Player jump to another scene
* GoToStartingPoint: in the case of gates / doors it indicates where to transport the Player. In the case of characters, I suggest using the "none" parameter. It makes no sense to make the Player jump to another scene
* NameOfInventory: What animation do I show in the inventory if I collect the character? By default indicate the value "none" because it makes no sense to collect a character

**Object**: please copy the whole object from "Obj\_PunkHead". Edit animations and object variables with this logic

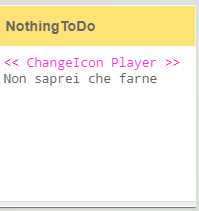
* KillOnTouch: Y/N . “Y” to kill on collide
* DialogueNode: Yarn node. It is triggered if the object is collected (for example when you collect the punk toupee)
* PickUpAble: Y/N If “Y” the Player can pickup it
* GoToScene: it changes the scene to the one indicated by the value of the parameter (for example a sign). For example, the S1L1 sign has parameter "L1S2"
* GoToStartingPoint: cambia la scena con quella indicata dal valore del parametro
* NameOfInventory: if it can be collected, it is the name of the inventory animation that will be shown. For example, the punk wig has the value **Inv1**. Attention due to technical limitations, the new items in the inventory must always have the name **Inv ..** + a number **Inv1**, **Inv2**,
* OtherAction = in the case of unforeseen actions (see example below) it is the identifier needed to understand which action to take

Dialogues:

The dialogues are all loaded into a **Yarn** file at the start of the scene. Depending on the language chosen, a different file is loaded.

Loading starts at this point. From here you can edit the file. Attention the file name must be the one indicated in the INTRO scene.

Pressing EDIT and modify with this logic:

For example: 

NothingToDo is the node name. It must be indicated when you want to start a dialogue with this text

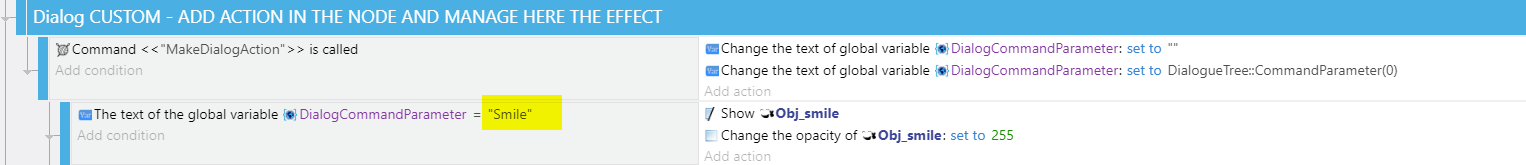
 is the command that makes the Player's face appear next to the text. This icon must be present in the animations of the object **DialogIcon**:

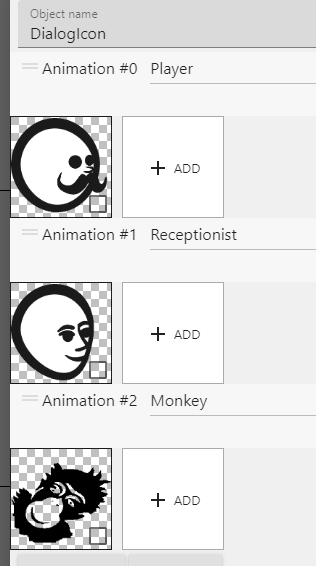
For example, insert this command in custom code area

to make the dialogue objects appear, their icon and the text "I don't know what to do with them"

In the dialog nodes it is possible to trigger other actions managed between commands. For example:

MakeActionSmiel is managed with the code:



Just add a new block and command in the node to handle this kind of need.

## FAQ:

*I want to insert a toupee object that can be collected*

It is NOT necessary to touch the code.

Let’s see **PunkHead** ( in Demo game). For a new object, add it to the Item group and BehindOfThePlayer (to see it behind) Modify the animation by changing the graphics

Let’s modify object parameters in this way

* KillOnTouch: N
* DialogueNode: none
* PickUpAble: Y
* GoToScene: none
* GoToStartingPoint: none
* NameOfInventory: Inv1 (inventory animation name)
* OtherAction = none

Please modify the **Inventory** object by adding two animations: Inv1 which represents the object and Inv1Selected which represents the selected object.

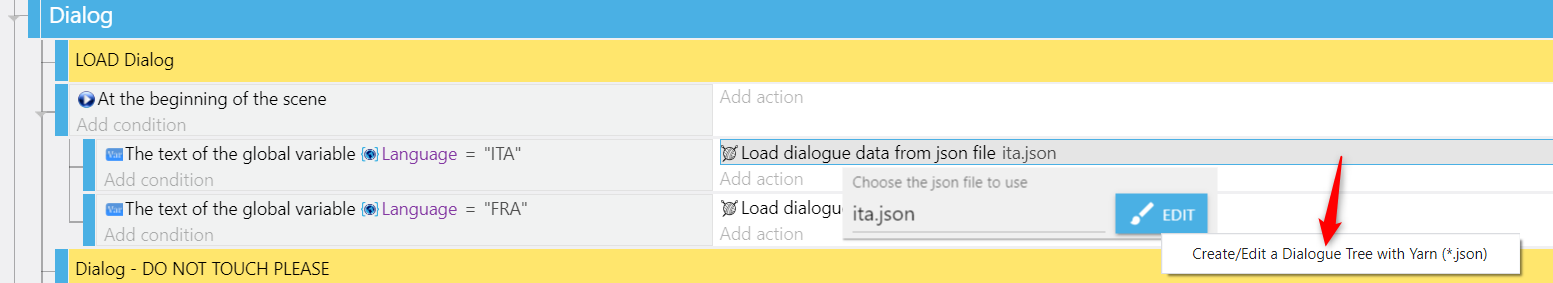
*create an item that can be collected and that displays a writing when collected*

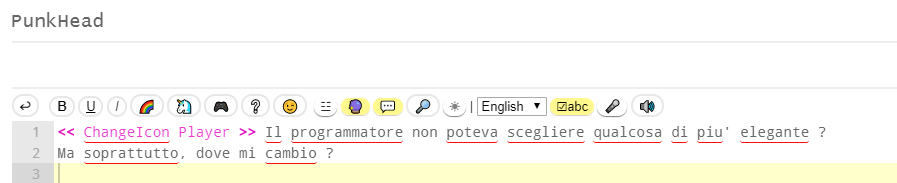
It is NOT necessary to touch the code.

Let’s copy PunkHead (from the demo game) and add it to the Item group and BehindOfThePlayer (to see it behind). I modify the animation.

* Let’s modify object parameters in this way
* KillOnTouch: N
* DialogueNode: PunkHead (when I pick it up it calls up the PunkHead node)
* PickUpAble: Y
* GoToScene: none
* GoToStartingPoint: none
* NameOfInventory: Inv4
* OtherAction = none

Modify the Inventory object by adding two animations: **Inv1** which represents the object and **Inv1Selected** which represents the selected object.

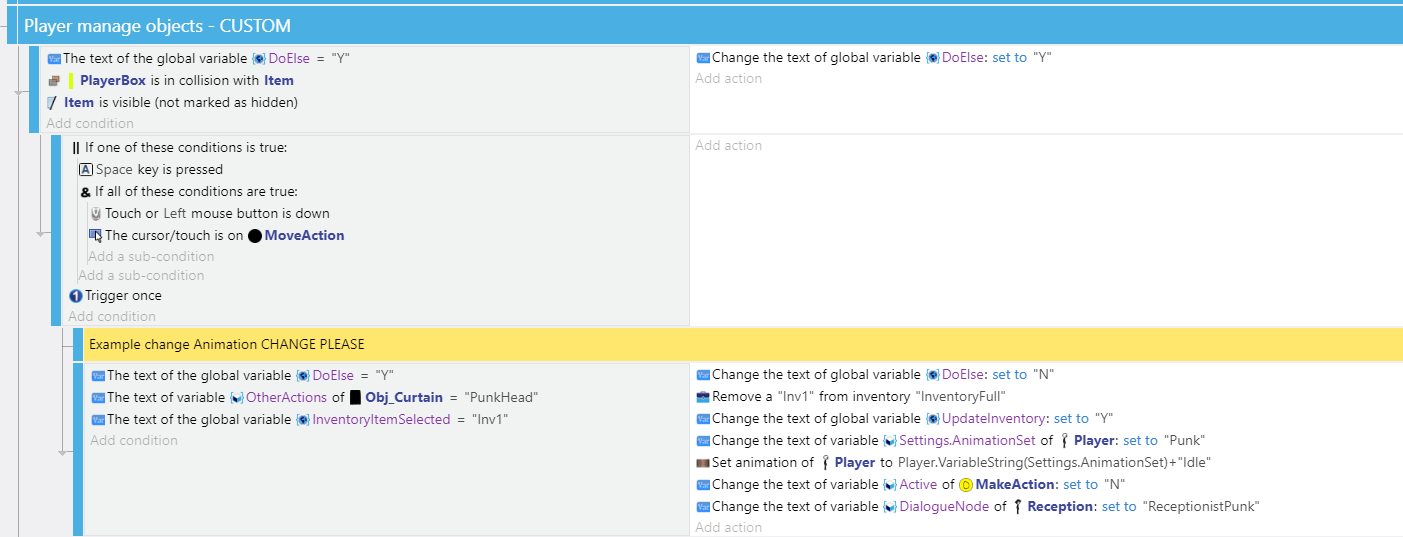
I open the dialog file and add a node named "**PunkHead**". It must begin with the indication of the Player icon and in text. For example:

Add a node called "**PunkHead**". For example

*<<ChangeIcon Player>> it means that when the message appears, the player icon will be shown alongside*

*Use an inventory item with one not collected to change the Player's animation (disguise)*

** Look for Example2 in the code**



….

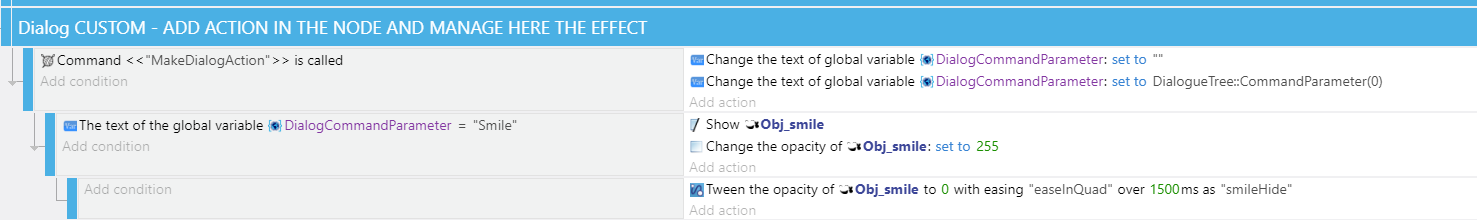
Change only

* The object selected in the inventory (InventoryItemSelected)
* The object to be used in the scene
* The name of the animation to be removed from the inventory (used object)
* The name of the disguise (Punk) that will define how to set up the animation

I want it to trigger a dialogue if I step in a certain point and show a sprite

** Look for Example1 in the code**

Add an Action button, with an animation (for example Act2) not yet used in the scene and add the following custom code



Change only what is yellow:

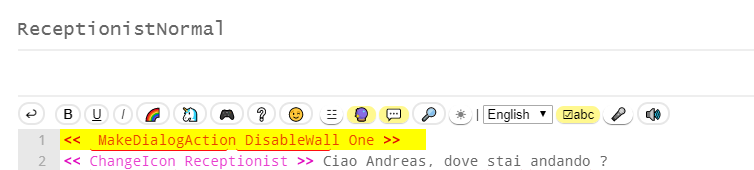
* MakeAction animation name (Act2)
* Dialogue node name (ReceptionistShop)

MagicWall deactivation during dialogue.

** Look for Example3 in the code**

Please enter the command in the dialog node

Insert a **MagicWall** and change the street name Name to "One"

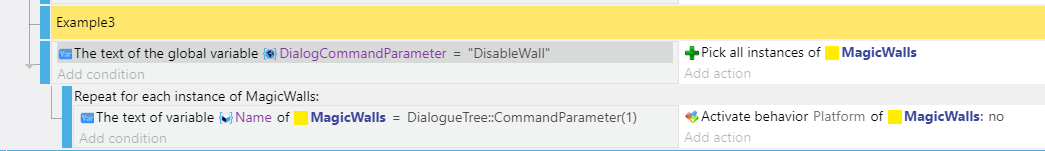
Insert the following node in the dialog and assign it to an Item

*Makeaction is the generic command to make an event happen*

*DisableWall represents the deactivation event*

*One MagicWall name to be disabled*





I want to add a neighboring column

Let’s create a new object (like the Mountains from the demo)

Add it to the Foreground group

Enter it n times of the game scene

*Add distant mountains*

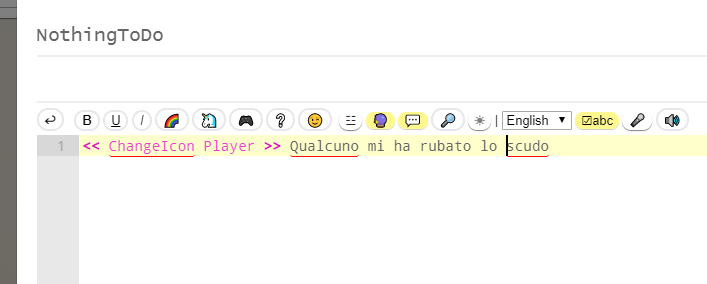
I create a new object (like the Mountains from the demo)

I add it to the Background group

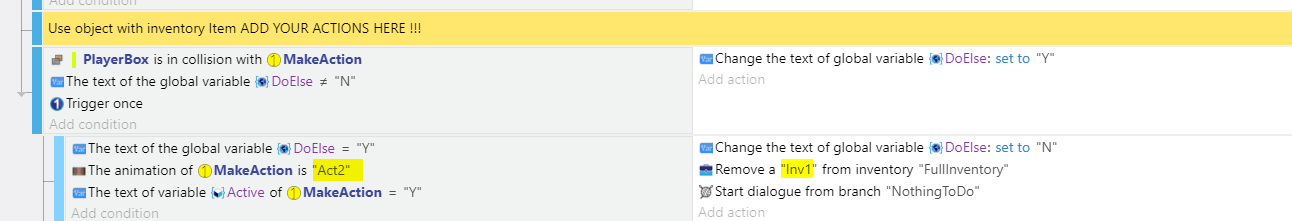
I enter it n times of the game scene

Remove an item from the inventory and force a message by passing through an action point

Create a new action point by associating an animation that has not been used as of now (e.g. Act2)

Create a new Node in the dialogue. For example

Please Enter the following code here:



##### The parts in yellow are to be changed:

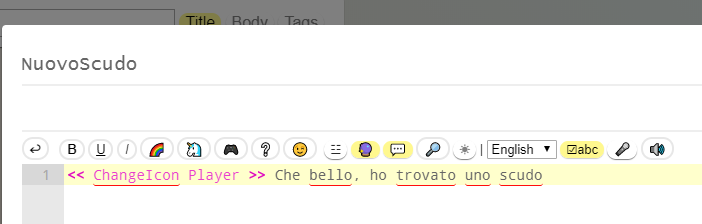
##### Act2 identifies the ActionPoint

##### Inv1 identifies the object to be removed from the inventory (Inventory Animation)

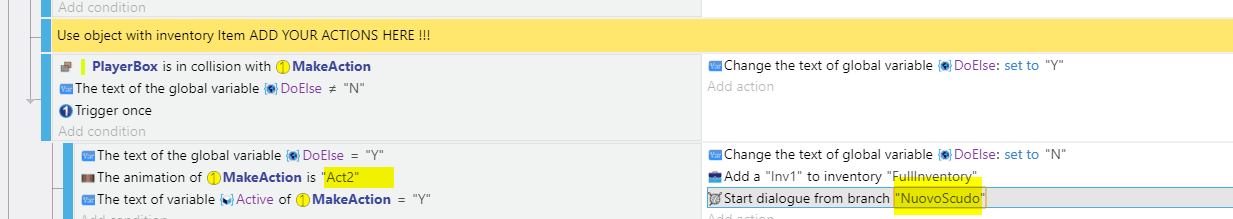
Add an item from the inventory and force a message by going through an action point

Create a new action point by associating an animation that has not been used as of now (e.g. Act2)

Create a new Node in the dialogue. For example



Enter here the following code here:



The only parts to change are in yellow: the animation that identifies the ActionPoint and the dialogue node

Global variables that it makes sense to modify

BackgroundSpeed background parallax speed

ForegroundSpeed Foregroun Parallax Speed

MoveParallax enable parallax

textScrollSpd: speed scroll dialogue

DebugMode enable display of debug objects

# DEMO Source

**The only points to customize are those in yellow.**