KOTLIN DECOMP

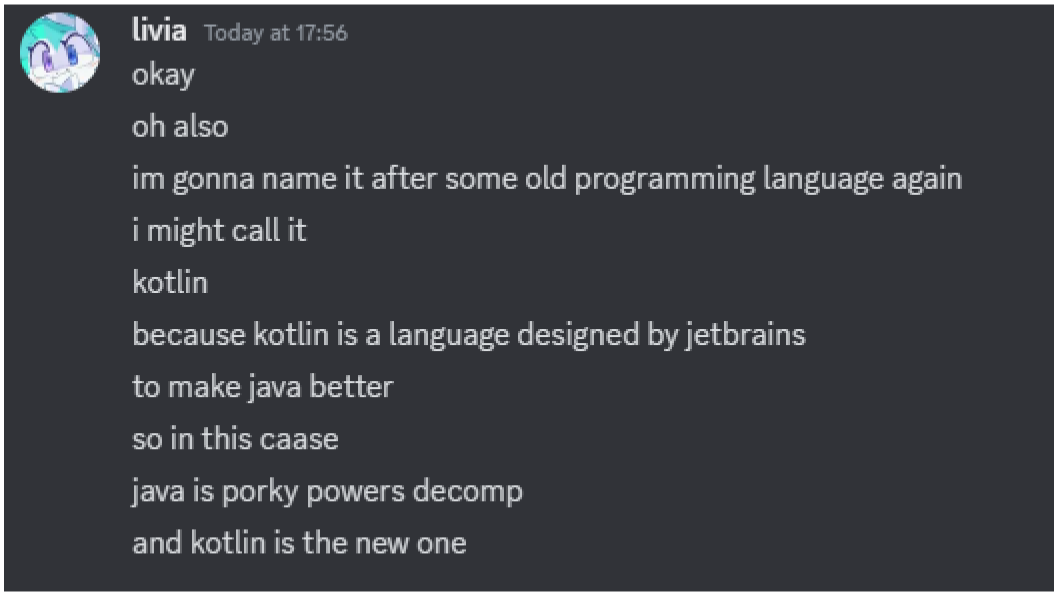
last updated 0.1 open alpha

TABLE OF CONTENTS

- A Brief History of Kotlin
- What Does Kotlin Package
- What Does Kotlin Change
- Kotlins API
- Contributors

A Brief History of Kotlin

Kotlin is PorkyPowers decomp on steroids!... and a decomp aimed at simplifying the Baldi modding process, by packaging your average tutorials into one big decomp, with handy customising features. The decomp was started on the 12th of July 2024 at around 17:50 (BST), and named after the programming language Kotlin, let past me explain:



livia Today at 17:56
okay
oh also
im gonna name it after some old programming language again
i might call it
kotlin
because kotlin is a language designed by jetbrains
to make java better
so in this caase
java is porky powers decomp
and kotlin is the new one

truly groundbreaking stuff

What Does Kotlin Package

Kotlin packages with 2 main assets

- Surge
- NaughtyAttributes
- Newtonsoft.JSON
- PrimeTween

These are all located in *Assets/_KOTLIN/Internal/Dependancies*

Surge is generally used internally for *Singletons*, and
NaughtyAttributes is used for *improving the Inspector*
Newtonsoft.JSON is used for *translation deserialization*
PrimeTween is used for *Plus elevator gates*

What Does Kotlin Change

Code

- Token comments are removed
- Tiny cleanups
- Organises scripts
- Items are structs, see KOTLIN.Items
- Interactions are inheritable, see KOTLIN.Interactions

Optimisation

- The map is converted to Quads
- Interactions are checked once a click, not a billion times
- Optimised Billboard & Pickup Animation scripts, see Contributors

Simplication

- Image text elements are replaced with text (for translations)
- Doors, Swing Doors & Windows are masks, see Contributors

Kotlins API

Kotlin has an API to make your life easier. Everything you need is documented below:

KOTLIN.Interactions

This is a class to *inherit* from, the GameController checks for an *Interactable* component on click and fires the *Interact* method, which you should override, for example:

```
public class InteractTest : KOTLIN.Interactions.Interactable
{
    public override void Interact()
    {
        UnityEngine.Debug.Log("wow ive been inteteracted");
    }
}
```

KOTLIN.Subtitles

This namespace handles *Subtitles*, all you need to know in this section is how to create a subtitle.

KOTLIN.Subtitles.SubtitleManager is a singleton, so you'll need SubtitleManager.Instance, then just call the CreateSubtitle method:

Arguments

SubtitleType type - 2D or 3D (or 4D but sets to 3D)

string text - what subtitle say

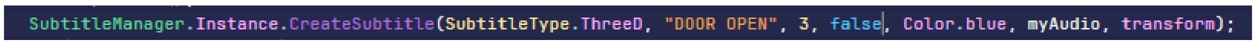
float time - how long on screen for

bool forever - should the subtitle stay on screen forever

Color colour - colour of subtitle text

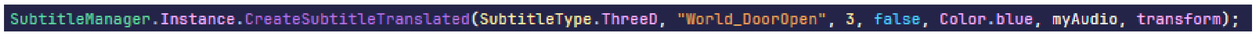
AudioSource audSource - what audio source created subtitle

Transform creator - what gameobject created subtitle



```
SubtitleManager.Instance.CreateSubtitle(SubtitleType.ThreeD, "DOOR OPEN", 3, false, Color.blue, myAudio, transform);
```

To translate a subtitle, use SubtitleManager.Instance.CreateSubtitleTranslated.



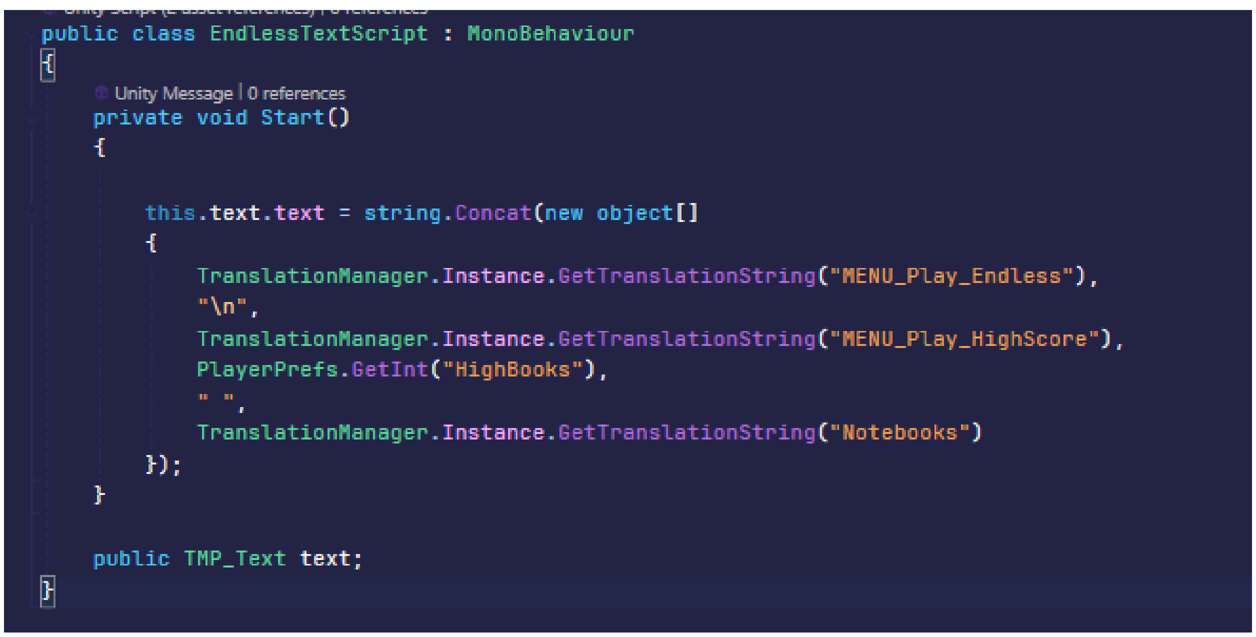
```
SubtitleManager.Instance.CreateSubtitleTranslated(SubtitleType.ThreeD, "World_DoorOpen", 3, false, Color.blue, myAudio, transform);
```

All arguments are the same, except text (argument 2) which should be the translation key, see the [Translation segment]() for more information on translations

KOTLIN.Translation

This namespace handles *translation*, translations are pretty simple.

To get a translation from a key, simply do TranslationManager.Instance.GetTranslationString("Key")
For example:

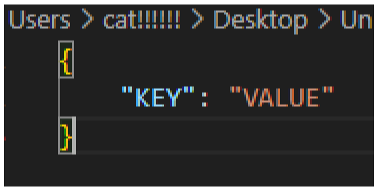


```
public class EndlessTextScript : MonoBehaviour
{
    Unity Message | 0 references
    private void Start()
    {

        this.text.text = string.Concat(new object[]
        {
            TranslationManager.Instance.GetTranslationString("MENU_Play_Endless"),
            "\n",
            TranslationManager.Instance.GetTranslationString("MENU_Play_HighScore"),
            PlayerPrefs.GetInt("HighBooks"),
            " ",
            TranslationManager.Instance.GetTranslationString("Notebooks")
        });
    }

    public TMP_Text text;
}
```

To create a translation, you need to create a JSON file in the StreamingAssets folder, and call it *Subtitles_{ANYTHING}*.json (file extension). Now add all your keys and stuff!1 It's kind of like a dictionary



```
{
    "KEY": "VALUE"
}
```

Value will show up in game

Translating Text

Add a *TranslationObject* to the gameobject and select what type of text yours is

TMPText is TMP text in the *World*

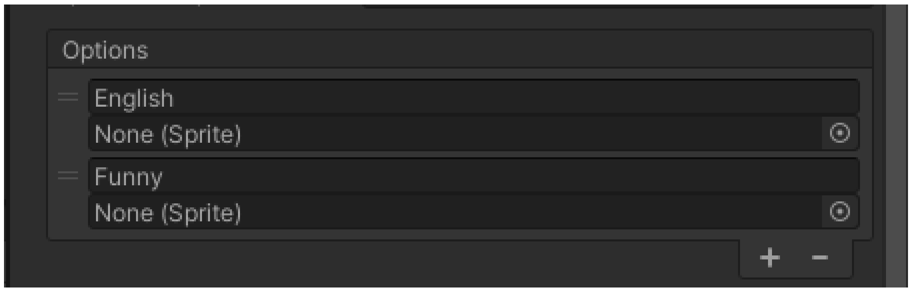
TMPText_UI is TMP text in *UI*

Text is unity's default text component

Then type in the key of the translation.

Adding translations to the options menu

In the *OptionsMenu* of the *MainMenu* scene, go to the *LanguageSelection* GameObject and scroll down to the *Dropdown* component. Find where it says *Options* and add a new entry, call it what you want the player to see.

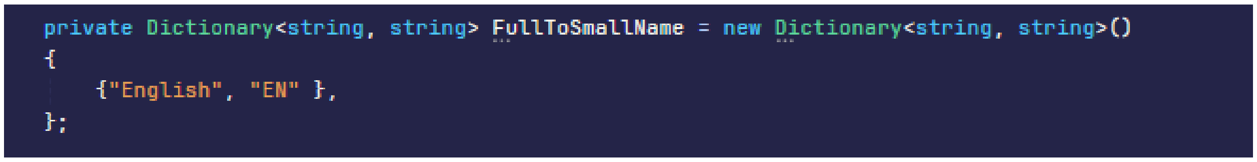


Options

English
None (Sprite)

Funny
None (Sprite)

Now open the *LanguageSelector* script and find the *FullToSmallName* dictionary.

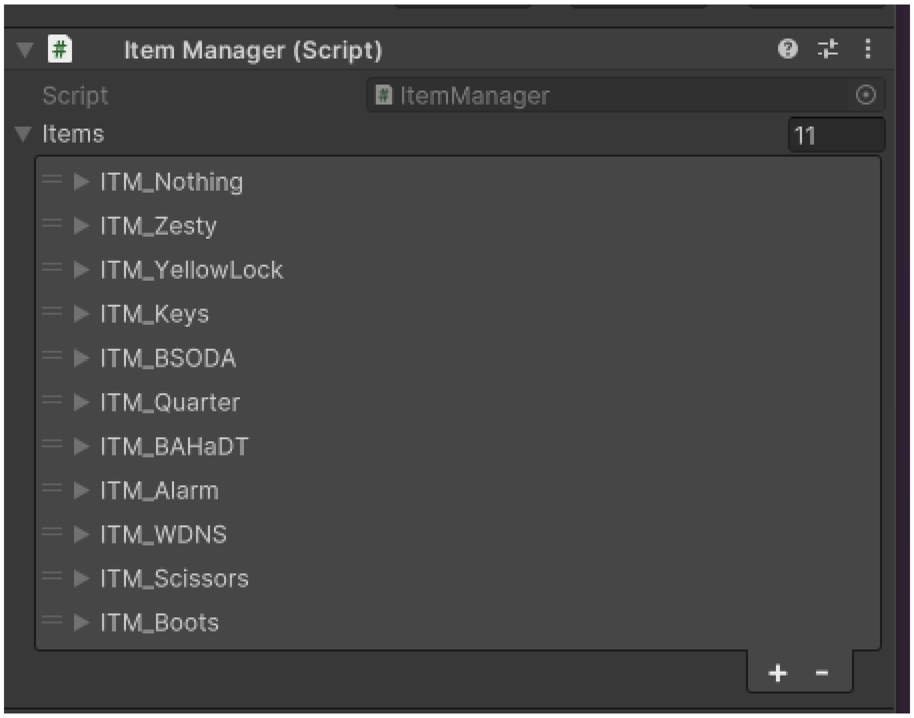


```
private Dictionary<string, string> FullToSmallName = new Dictionary<string, string>()
{
    {"English", "EN" },
};
```

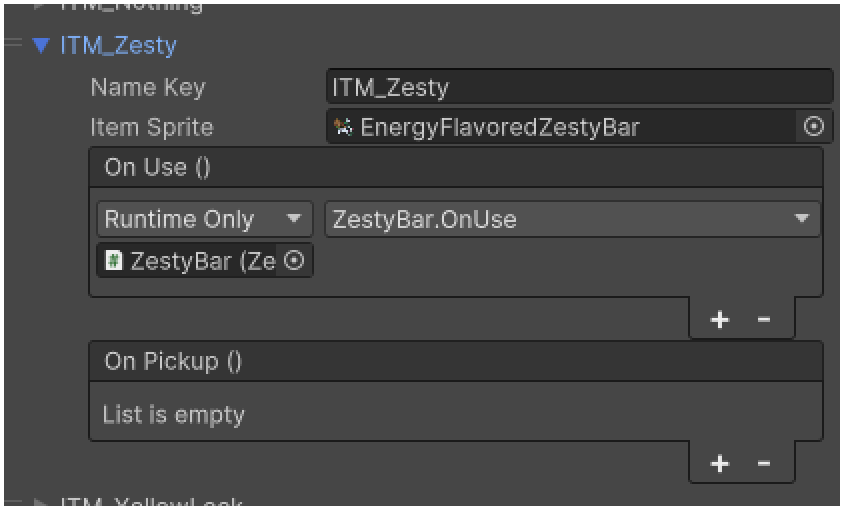
Add a new entry to the dictionary, with the key (first string) being what you inputted in the dropdown, then the value (second string) being what you called the subtitle file identifier (so for Subtitles_EN.json you would put EN)

KOTLIN.Items

Uhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
Each GameController needs an *ItemManager* component, which manages each item's data *(1.0 should replace all item functionality into the ItemManager)*

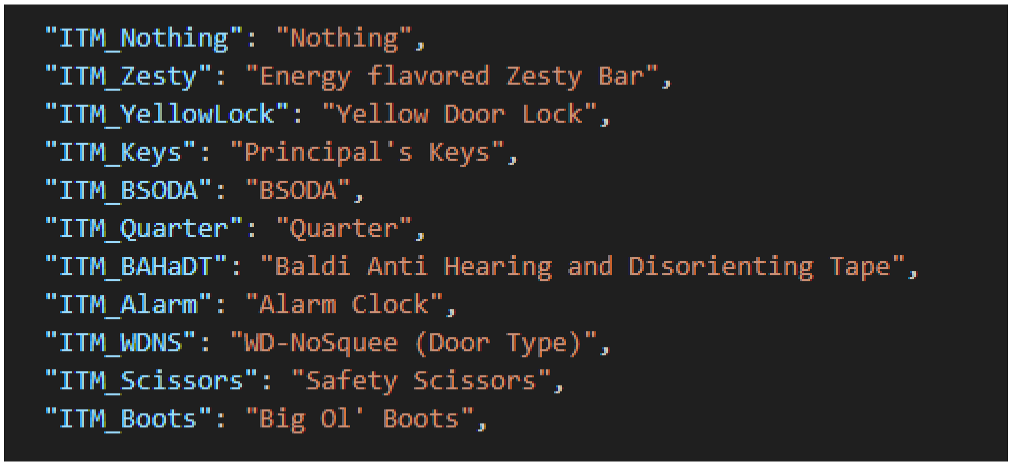


The ItemManager has a list of every item and it's data, click on an arrow to expand the dropdown and show it's data



Name Key: ITM_Zesty
Item Sprite: EnergyFlavoredZestyBar
On Use (): Runtime Only, ZestyBar.OnUse

Name Key is the **translation key** for the items name, for example:



```
"ITM_Nothing": "Nothing",
"ITM_Zesty": "Energy flavored Zesty Bar",
"ITM_YellowLock": "Yellow Door Lock",
"ITM_Keys": "Principal's Keys",
"ITM_BSODA": "BSODA",
"ITM_Quarter": "Quarter",
"ITM_BAHaDT": "Baldi Anti Hearing and Disorienting Tape",
"ITM_Alarm": "Alarm Clock",
"ITM_WDNS": "WD-NoSquee (Door Type)",
"ITM_Scissors": "Safety Scissors",
"ITM_Boots": "Big Ol' Boots",
```

Item Sprite is the.. well.. **Sprite of the item**.. obviously

Then, you'll see 2 UnityEvents; *OnUse & OnPickup*
What they do is self explanatory really. Just read up on UnityEvents to understand how they work.
You can use them to do literally whatever you want *so long as the method only has 1 or less argument lmao*

You'll notice how Kotlin uses its events, each item is a GameObject as a child to the ItemManager with the respected components attached, then UnityEvents linking them. This is the recommended and cleanest way to do it.

KOTLIN.Lighting

Kotlin uses shader-based tile lighting, meaning you require a Lightmap to light your scenes. To change the current global lightmap at runtime, you can do
LightingManager.Instance.UpdateLights(Texture2d yourMap)
Note that **yourMap is an optional parameter**, you can do
LightingManager.Instance.Lightmap = yourMap
then call UpdateLights with a null parameter, if it suits you more.

Upon opening the decomp for the first time, the map will be dark, you must go to KOTLIN > Fix Lighting, input the lightmapand click apply to light your map up.

KOTLIN.Events

This is super simple, and everything is self explanatory, but basically you have an event manager, create an empty GameObject childed to the EventManager's GameObject and attach the event script to the child. The manager will automatically find the script upon runtime. **You must make sure the DoCountdown bool is true, KOTLIN defaults this to false as there is no builtin events, you can find it in Game > EventManager**.

To code an event, create a new script and inherit from *RandomEvent*. Override *StartEvent* to setup and override *EndEvent* to run code when ur event ends (just make sure to call the method lmao), **make sure to do base.EndEvent() at the end of your override**.

Contributors

- BlueVapor1234 - , Plane 2 Quad Convertor (Edited), Subtitle Position & Scale Calculation
- YuraSuper2048 - Optimized Billboards, Optimized Pickup Animation
- Benefond - Polish Translations
- PogoDev - Tile Lighting help :D