Documentation

Welcome to the documentation for custom levels on Jump King using JumpKingPlus! On your left you can find the table of contents with everything you should need to make a custom level.

Warning

When this document referres on a folder, this is obviously meant to be inside `Jump King/Content/mods`, JumpKingPlus does **not** take any responsibility of your actions!

I reccomend you to use the **working sample level** to start with, and edit it for your new custom level!

Requirements for building custom levels

> Custom levels are available only using **JumpKingPlus on v1.2.0 or above**.

- The **sample custom level** by Phoenixx19
- A simple pixel art editor (Aseprite or GraphicsGale)
- A good image editor for editing and exporting hitboxes (GIMP or Adobe Photoshop)
- JumpKingManager to access one area quickly
- Programs to convert images, music and fonts:
  - XNBCLI for converting **images** into XNB and viceversa
  - Fast XNB Builder for converting **images and music** into XNB
  - Visual Studio 2019 (or above) using MonoGame for converting all files; check out more **here**, this is pretty long and time consuming I suggest you to not use it unless you have to.

Common rules

Level design in Jump King is a delicate balance between fairness and hardness. These rules are not only made to prevent unfair and impossible levels but to respect Nexile's original ideas on level design. Also in order to get your map approved on the site, these rules **need** to be followed.

1. **Screen transitions must be full jumps**; it would be unfair for a player not knowing how to jump over a new screen

2. **Transition platforms should always work** (with a full jump) and they **must not be related to a specific position** in the platform before the transition to the new screen
3. **Platforms must be bordered with a line** (with at least 1px)
4. Do not exaggerate with the Lost Frontier jumps (8px platform equals to 1px in the hitboxes file), that area sucks

Testing is the most important phase of your level that should take you a lot of time, a good level has every single fall calculated, nothing is left to be random. Check out some more tips **here**.

Testing

As said before, testing is a very important part of creating a custom level. In order to get your files working in-game; here's a section dedicated for that.

Getting started

0. Download the sample level from the requirements.
1. Drag the `mods` folder from the zip file to `Jump King/Content` folder.
2. You now have a custom map ready to work on!

Convert images and music (Fast XNB Builder)

Fast XNB Builder can only pack images.

1. Download the release for Fast XNB Builder in the links above.
2. Create a folder with all the items you want to pack.
3. Open `Fast XNB Builder.exe` and select the folder you previously created.
4. If succeded, you will find your files inside `/Final` folder.

Convert images (XNBCLI)

0. Install prerequisites of XNBCLI available here.
1. Download the latest release for XNBCLI in the links above.
2. Export the `xnbcli-windows-x64.zip` file.

Unpack images

If you want to unpack an image from the game, put the file inside the `packed` folder and open the `unpack.bat`. If succeeded, you will find your files inside the `unpacked` folder.

Pack images

If you want to pack an image to put on the mod, make sure you have the .json file of your file ready to get packed with your image. If you never unpacked an image you can use this simple .json and modify for your own use!

YOURFILENAMEHERE.json

```
{
    "header": {
        "target": "w",
        "formatVersion": 5,
        "hidef": false,
        "compressed": false
    },
    "readers": [
        {
            "type":
"Microsoft.Xna.Framework.Content.Texture2DReader",
            "version": 0
        }
    ],
    "content": {
        "format": 0,
        "export": "YOURFILENAMEHERE.png"
    }
}
```

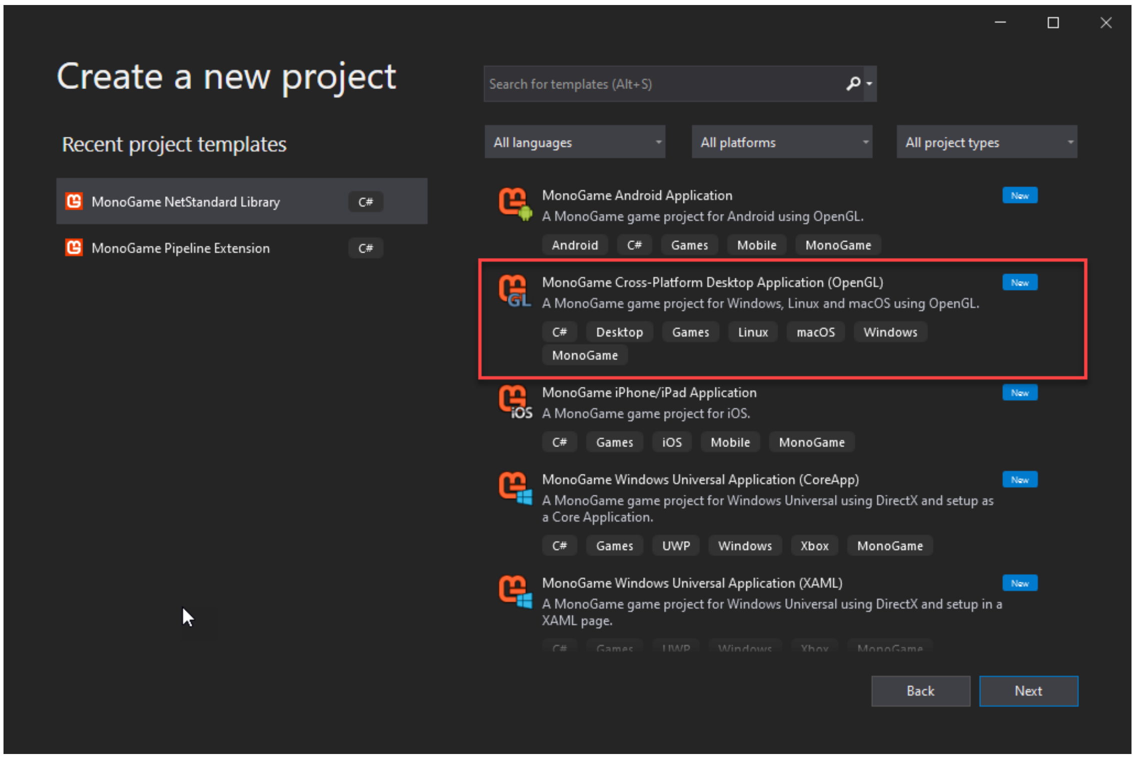
Once put your image inside the `unpacked` folder with the .json file, you can open the `pack.bat`. And if succeeded, you managed to create your very own custom texture! The packed file can be found inside the `packed` folder.

Convert all (VS2019+MonoGame)

Last warning. This takes a lot in both space on your drive and time. Choose only if this is your last hope.

1. Install Visual Studio Community 2019.
2. Download *.NET desktop development*.
3. Install MonoGame from the ufficial website.
4. Follow these instructions (from **Install MonoGame extension for Visual Studio 2019** to **Install MGCB Editor** included!).

5. Reopen and create a new project in Visual Studio with the template:



MonoGame Android Application
A MonoGame game project for Android using OpenGL.
Android C# Games Mobile MonoGame

MonoGame Cross-Platform Desktop Application (OpenGL)
A MonoGame game project for Windows, Linux and macOS using OpenGL.
C# Desktop Games Linux macOS Windows MonoGame

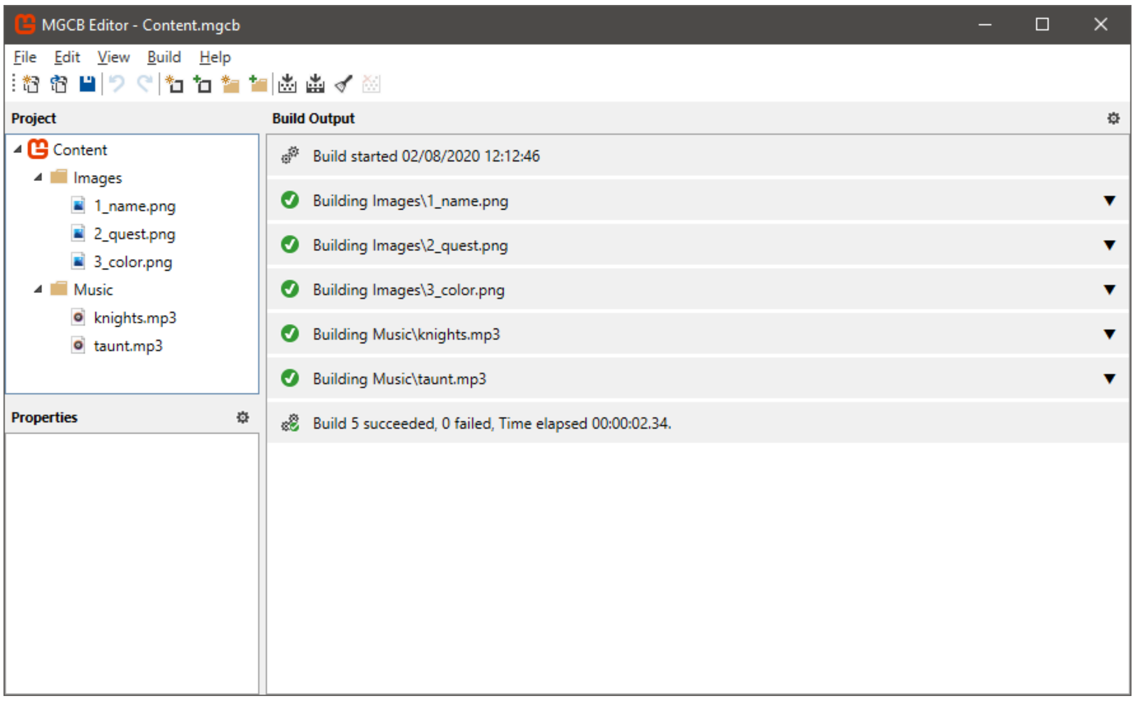
MonoGame iPhone/iPad Application
A MonoGame game project for iOS.
C# Games iOS Mobile MonoGame

MonoGame Windows Universal Application (CoreApp)
A MonoGame game project for Windows Universal using DirectX and setup as a Core Application.
C# Games UWP Windows Xbox MonoGame

MonoGame Windows Universal Application (XAML)
A MonoGame game project for Windows Universal using DirectX and setup in a XAML page.
C# Games UWP Windows Xbox MonoGame

(the project name does not matter)

6. On the right side of the screen (Solution Explorer), open the folder Content and open the file `Content.mgcb`. If you have installed MGCB Editor this will show up:



```
Build started 02/08/2020 12:12:46
Building Images\1_name.png
Building Images\2_quest.png
Building Images\3_color.png
Building Music\knights.mp3
Building Music\taunt.mp3
Build 5 succeeded, 0 failed, Time elapsed 00:00:02.34.
```

7. To add convert images, audio files and fonts you need to add files using the rectangle with a yellow asterisk with tool tip saying Add Item.
8. Select all the items you have to convert. To convert fonts, **follow these instructions**.
9. Save the Content file and build it.
10. You will find all your packed file in: `project directory/bin/x86/Debug/Content/`.

Modding

After installing and downloading all the files needed; you can start working on your first custom level. In order to make the custom level working you will need to create two files inside your `JumpKing/Contents/mods` folder. JumpKingPlus loads the custom mode when both the `level.xnb` and the `mod.xml` files are in the folder above.

Mod config file

Blank mod.xml Example mod.xml

In this file, you will set up the basics information of your level such as:

name	about	optional
`<About>`	Contains the fundamentals of the mod	✕
`<title>`	Title of the custom level (will show up in the Stats Display window)	✕
`<ending_screen>`	Screen where the babe spawns	✕
`<Fonts>`	Array of available fonts (MenuFont, MenuFontSmall, StyleFont, OptimusUnderline, Tangerine, LocationFont, GargoyleFont)	✓
`<Ending>`	Contains the babe ending images, only one story is available for now	✕
`<MainBabe>`	Screen for beating the custom game	✕
`<MainShoes>`	Screen for beating the custom game with the Giant Boots	✕
`<EndingLines>`	Credit[] (or array of credit)	✓
`<Credit>`	Contains the header and the strings for the ending lines	✓
`<header>`	Header for ending lines	✓
`<People>`	Array of strings for ending lines	✓
`<string>`	Ending lines (from 1 to 5 works fine)	✓

The title and the ending screen are necessary to make the custom level playable. The title will show up only when the game started is Main Babe / Normal Game.

Custom fonts are optional as specified, if left to blank, JumpKingPlus will automatically pick the default ones.

Both MainBabe and MainShoes are supposed to be called as the .xnb files located into `/mods/ending`. **Do not** include the extensions in the name, the game is automatically set to find the .xnb files.

Inside the Ending Lines, it's possible to use the default library for translations included in the game, LanguageJK.

Hitbox file

The hitbox file is a **Texture2D** (.png image with alpha channel or transparent) with the size of 780x585 pixels. Every screen is ordered by column starting from top to bottom which means every screen has 60x45 pixels. Jump King uses a specific color to define what a block is inside of this file.

block	description	usage	color
Solid	A normal block you can stand on		⬛
Slope	The player will slide standing on it	Needs two adjacent blocks	🟥
Fake	The player will fall through it	Wind, water and low gravity affects it	▨
Ice	The player will slide on it but can still stand on it		🟦
Snow	The player will remain at their position unless trying with another jump	Snake Ring bypasses it	🟨
Wind	If placed on a screen, it will slide the player slowly to a direction	The wind polarity reverses every 5 seconds	🟩
Sand	The player will slowly fall through the block, while jumping and walking is still possible		🟧
No Wind	The player will fall through it	Wind, water and low gravity does not affect it	⬜
Water	Velocity and gravity is halved		🟦
Quark	Rounds the player's Y position to make falls less different; reference [here]	Used when player is in full velocity	🟩
Teleport	If placed on a screen, it teleports the player to a specific screen using the **RED** of the RGB as the screen number	Works both left and right side of the screen	🟦🟪

block	description	usage	color
Low gravity	Velocity and gravity is between water and normal, distance is slightly higher		🟦

The yellowish color defines a custom hitbox added with JumpKingPlus.

Screens folder

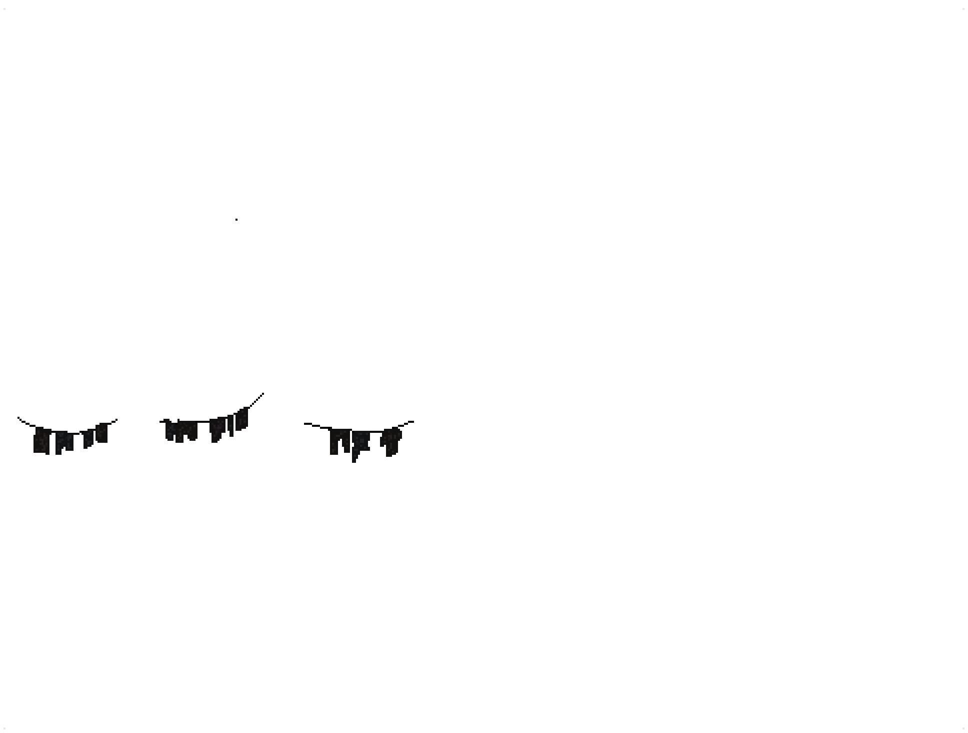
The screen folder contains textures such as background, foreground, midground, scrolling images and masks.

Background

The background is usually used for skies or gradients to put back on a certain or multiple screens.
The name of the file should be `bg(SCREEN NUMBER).xnb`, or as an example, `bg1.xnb`.

Foreground

The foreground is used for details that are in front of the player, such as vines or grass.
The name of the file should be `fg(SCREEN NUMBER).xnb`, or as an example, `fg1.xnb`.

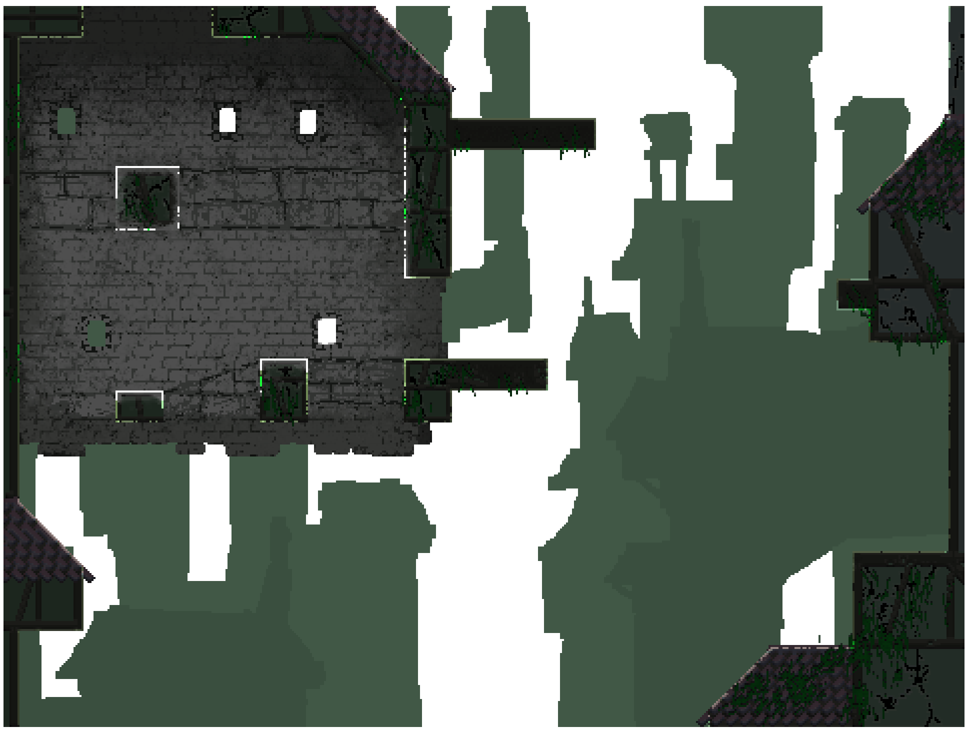


Masks

The masks are animated backgrounds that are stored inside the default `particles` folder. Masks can be used to give more depth to the level, some examples of masks are ash, rain and snow.
The name of the file should be `(MASK NAME)mask(SCREEN NUMBER).xnb`, or as an example `light_snow_bgmask1.xnb`.

Midground

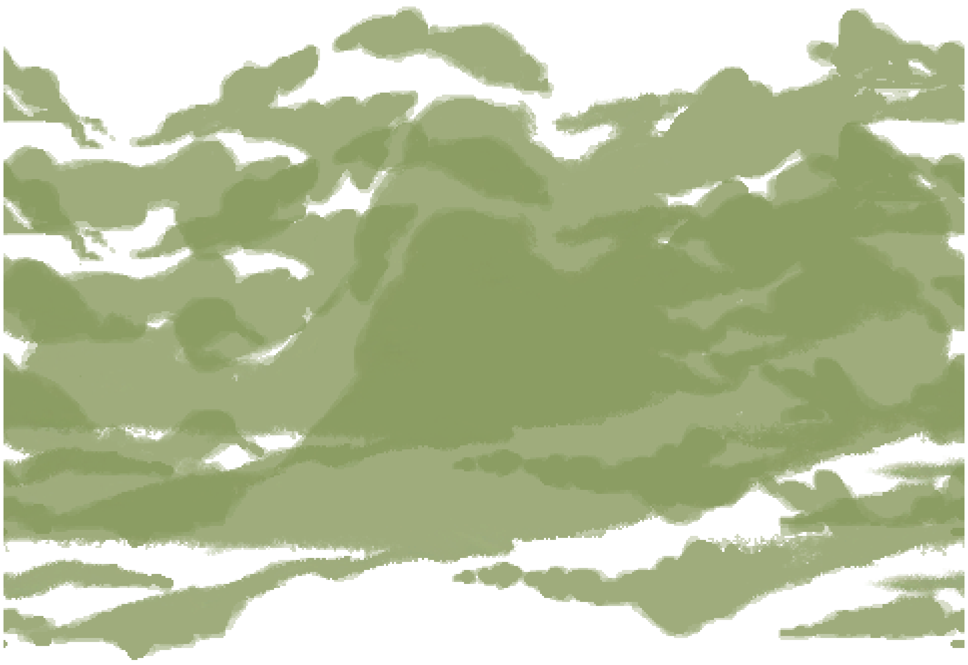
The midground is usually used for platforms and detail that want to be behind the player (the player can go over them).
The name of the file should be `(SCREEN NUMBER).xnb`, or as an example, `1.xnb`.



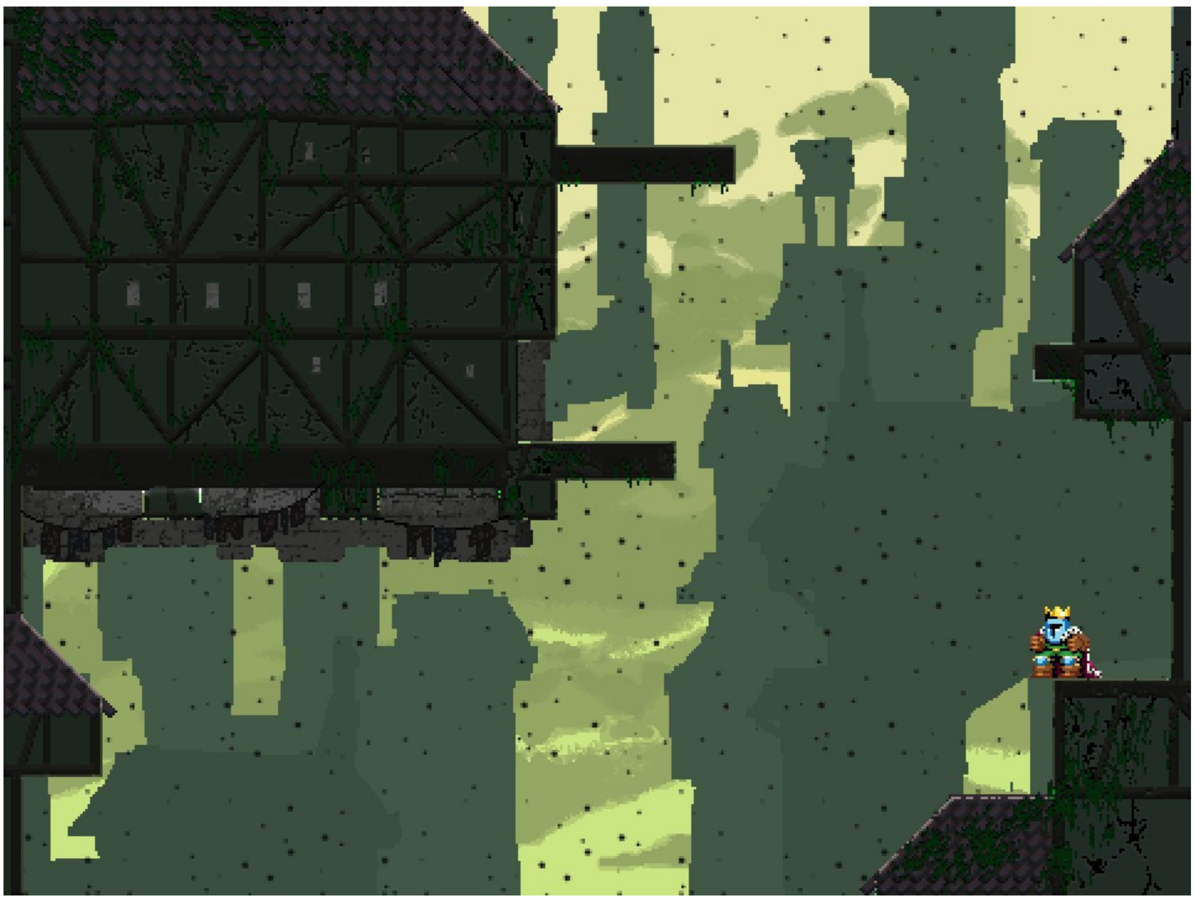
Scrolling images

Example scroll.xml

The scrolling images are managed by an .xml file, that determines their texture, position, velocity and layer mode (see example scroll.xml above). The scrolling texture is usually used for clouds or birds flying in the distance. The texture name should be the same of the namefile. Which means if you created a new scrolling texture called `clouds.xnb` the name of the texture inside the scroll setting file should be `<texture>clouds</texture>`.



All of the layers together make this (not counting the hidden wall because that's a prop):



Props folder

The props folder contains textures and settings of props used in-game; their categories are: worlditems, textures, messages, hidden walls and hidden walls props. Avoid using props in the final screen to prevent slight visual bugs from the game itself.

World items

Example worlditems.xml

The world items are items that the player can pick up by walking on them. These have their own texture from the wearable items and they are stored in `props/worlditems`.

The game to position and read their texture reads a configuration file called `worlditems.xml`. The file is self explainatory so there's no need of a table.

NPCs

Everything related to NPCs can be found inside `textures/old_man`. NPCs can be of two types: which are `old man` (normal NPC that speaks only) and `merchant` which is an entity that includes the `old man` type but can also sell items. The textures for both needs to be inside `textures/old_man` while the quotes work differently.

old_man_quotes.xml file (located in `textures/old_man/lines`)

tag	description
`<name>`	Name of NPC (LEAVE AS DEFAULT)
`<sheet_cells>`	Size of the spritesheet
`<X>`	Columns
`<Y>`	Rows
`<random_count>`	(LEAVE AS DEFAULT)
`<position>`	Position inside the screen
`<X>`	Position on X axis
`<Y>`	Position on Y axis
`<home_screen>`	Screen number
`<talk_animation>`	Contains frames, triggered when talking
`<frames>`	AnimationFrame[]
`<AnimationFrame>`	Contains sprite index and duration
`<sprite_index>`	Index of sprite from sheet cells
`<duration>`	Duration in seconds
`<busy_animation>`	Contains frames, triggered when idle
`<bubble_format>`	Format regarding text
`<direction>`	Left or right
`<anchor>`	Contains X and Y
`<X>`	Left or right
`<Y>`	Up or down
`<width>`	Width in pixels
`<trigger_box>`	Box where NPC starts speaking if player is inside
`<center_x>`	Leave as defualt unless you know what you are doing
`<center_y>`	Leave as defualt unless you know what you are doing

tag	description
`<width>`	Width in pixels
`<height>`	Height in pixels
`<talk_delay>`	Delay of before starting to talk
`<talking_speed>`	Speed until new letter reveal (in seconds)
`<intro_quote>`	Contains intro quotes or lines
`<lines>`	string[]
`<string>`	A single quote
`<achievements>` and inside tags	Leave as default
`<screens>`	OldManScreen[]
`<OldManScreen>`	Contains all the information to unlock a new quote
`<screen>`	Screen number
`<quotes>`	OldManQuote[]
`<OldManQuote>`	Contains intro quotes or lines

> To make this table have more sense, look inside your gamefiles and look for an example (`Jump King/Content/props/textures/old_man/lines/hermit_quotes.xml`), this will clear out more than watching a table.

merchant_quotes.xml file (located in `textures/old_man/merchant`)

tag	description
`<sale_item>`	Item being sold
`<currency_type>`	Item that serves for payment
`<price_increase>`	Price
`<easteregg_amount>`	Easter egg amount (JumpKingPlus Exclusive)
`<sale_achievement>`	Leave as default
`<sale_lines>`, `<easteregg_lines>`, `<sold_lines>`, `<no_gold_lines>`	OldManQuote[]
`<sell_quote>`	Contains lines
`<lines>`	string[]
`<settings>`	Old Man settings (the table above)

Raven

The raven is the entity related to the bird. It is located inside the `textures/raven` folder. The folder should contain the raver texture and the `(raven name).ravset` (which is a xml file).

tag	description
`<fly_sfx>`	Sound effect that needs to be played when the flight is triggered
`<texture>`	Name of the raven (which should be the same name of the file)
`<item>`	Items carrying
`<positions>`	RavenPosition[]
`<RavenPosition>`	Contains the positions of the raven
`<screen>`	Screen number
`<position>`	Contains X and Y
`<X>`	Pretty self-explainatory
`<Y>`	Pretty self-explainatory
`<treasure>`	Boolean value. If true, the raven is carrying the item
`<look_direction>`	Left or right
`<fly_direction>`	Left or right

Props

Example prop_settings.xml Example prop.xml

The props in-game (such as the bonfire in the first screen) are stored in the `props/textures` and they have one setting file named `prop_settings.xml` which contains:

name	description
`<settings>`	PropSetting[]
`<PropSetting>`	Contains settings of a single prop
`<name>`	Name of the file
`<fps>`	Frames per second
`<frames>`	float[]
`<float>`	Time per frame
`<sheet_cells>`	Size of the spritesheet
`<X>`	Columns
`<Y>`	Rows
`<random_offset>`	Optional tag to get random offsets

To add a prop on a screen, you will need to create a configuration file called `prop(SCREEN NUMBER).xml` and add each prop with their type (name of the prop), position on X and Y axis (**0,0 is top-left!**) and optional if the prop should be flipped.

Hidden walls

Example hidden_wall.xml Example prop.xml

Hidden walls are used in-game to hide areas or make the screen more realistic, the hidden wall works as foreground until the player gets into its position where it gets transparent.

The hidden walls are located in `props/hidden_walls/textures` and they are managed by a configuration file called `hidden_wall(SCREEN NUMBER).xml`.

Hidden walls can have props too; these can be added creating a different prop configuration file inside `props/hidden walls props` called `prop(SCREEN NUMBER).xml`.

King folder

The king folder contains the textures of the wearable items by the player, by changing these you will have a different texture for the item only when the custom mode is triggered. Keep the same item name in order to get it working.

Locations

Example location_settings.xml

The locations in-game can be changed using the `gui/location_settings.xml` file.

tag	description
`<locations>`	Location[] or array of locations
`<Location>`	Location information
`<start>`	Screen number where location starts
`<end>`	Screen number where location ends
`<unlock>`	Screen number where location name pops up
`<name>`	Location name

Font folder

The font folder should include the custom fonts included for the custom level. Sadly MonoGame, and so Jump King, does not support TrueType fonts (.TTF) so to make them compatible in-game, you should check out how to convert the font to make it compatibile below **here**.

Audio folder

The audio folder should contain all the possible sound related content such as: background (ambiental music or music of a specific zone), music (ending songs) and event effects sounds.

Ambient as background

The ambient music should be placed inside `audio/background` as an .xnb file. To add this to a specific zone, simply edit the `audio/background/data/values.xml` and add on the AmbienceSave section like so:

```
<sections>
  <AmbienceSave>
    <ambience>
      <Ambience>
        <name>Water</name>
        <volume>0.7</volume>
      </Ambience>
      <!-- more Ambience -->
    </ambience>
  </AmbienceSave>
  <!-- more AmbienceSaves -->

  <screens>5</screens>
</sections>
```

values.xml file

Example values.xml

tag	description
`<special_info>`	AmbienceInfo[]
`<AmbienceInfo>`	Contains all the information about a music piece
`<name>`	Name of the music file
`<type>`	Should be always `Music`
`<restart>`	Boolean value. If true, the song will be looped
`<fade_out_length>`	Fade out length in seconds
`<fade_in_length>`	Fade in length in seconds
`<sections>`	AmbienceSave[]
`<AmbienceSave>`	Contains a section with sounds
`<ambience>`	Ambience[]
`<Ambience>`	Contains name and volume of a sounds
`<name>`	Name of the file
`<volume>`	Volume from 0 to 1 (basically 0-100%)
`<screens>`	Amount of screens for sections

WARNING: The `<screens>` tag doesnt work with the name logic of the screen number. This is precisely made to avoid mixmatches.
From the first `<AmbienceSave>` it keeps the `<screens>` number counting.
So if you have two AmbienceSaves where the first one has 5 screens and the

second has 2, you would have done already 7 screens of music.

Music as background

Following the name concept of the above, this shares same folder but has to be configurated differently. In the `values.xml` file mentioned above, this needs to be configured as an AmbienceInfo as well.

```
<special_info>
  <AmbienceInfo>
    <name>TheBogTwo</name>
    <type>Music</type>
    <restart>true</restart>
  </AmbienceInfo>
  <!-- more AmbienceInfos -->
</special_info>

<sections>
  <AmbienceSave>
    <ambience>
      <Ambience>
        <name>TheBogTwo</name>
        <volume>0.85</volume>
      </Ambience>
      <!-- more Ambience -->
    </ambience>
  </AmbienceSave>
  <!-- more AmbienceSaves -->

  <screens>5</screens>
</sections>
```

Music

The music contains many other files that can be changed such as:

- Main babe ending song (optional)
- Menu intro (optional)

And event music which is a subfolder which contains music that can be triggered, such as the sound on the last babe screen or the gargoyles. There is a very easy configuration file (`audio/music/event_music/events.xml`) that does not need any explaination.

Post Scriptum: This folder has been cut to contain custom levels' size, since the audio folder takes up 80% of the size.

Particles folder

Example weather.xml Example snow_settings.xml

The particles folder should include two .xml configuration file which are:

- snow_settings.xml (which sets the snow)
- weather.xml (which sets the weather)

Weather

tag	description
`<weathers>`	Weather[]
`<Weather>`	Contains name of the weather, fps and screens it is in
`<name>`	Name of the weather (leave as default)
`<fps>`	Frames per second
`<screens>`	int[]
`<int>`	Screen number

By setting the weather to a specific screen number, that screen also can use masks to crop the particles at will.

Snow settings

This needs to be configured, if the snow particle is used.

Publishing

To get it published on the site, post your map in the #modding channel on Discord where it will get "verified" by the players. The zip file should contain your mods folder and only the files needed for the level custom.

~Phoenixx19, 2020