For Developers

Packaging

The MFB package is made up of:

- Blender 2.69 64bit Binary for Mac and Windows
  - blender - Self-compiled from source (XCode 5; Visual Studio 10)
  - blenderplayer - self-compiled from source, same as above
  - Python 3.3 - which is included in Blender

> On the Mac, `blender` and `blenderplayer` are manually combined into one app bundle to avoid having two identical copies of Python. On Windows, this step is not needed.

- Additional Python Libs that needs to be installed manually into the Blender Python folder:
  - `tkinter` for Python 3.3
  - `unittest` for Python 3.3
  - `numpy` 64bit for Pyton 3.3
    - Mac: Get `numpy-master` from github, run `python setup.py`, then select `build`
    - Windows: Get numpy 64bit for Py33 from `http://www.lfd.uci.edu/~gohlke/pythonlibs/`
- MGLTools (Python code)

  - OS agnostic
  - Provided by MGL of Scripp Institute
  - Updated with `2to3` on selective files
  - Updates with some manual patching for Python 3 support (Type check, strings)

- MGLTools Binary packages: (MSMS, UTPackages)

  - OS dependent
  - provided by MGL of Scripp Institute
  - compiled specifically for Python 3.3 64bit

- Additional Python code

  - Mostly interfaces between the Blender game engine and the MGLTools.

- Additional asset, textures, etc

Setting up Development Environment

1. Acquire everything in the aforementioned MFB package (Blender, MGLToolKit, code, etc)
2. Compile Blender for target OS
3. Acquire and install Numpy to Blender Python directory
4. Now you are ready to develop!

Making a release

1. Set settings.useDebug to False
2. Set appinfo.AppVersion to reflection version change
3. Set info.plist to reflect version change (applies only to Mac releases)
4. Run `python3 builder.py` script to create a release

The `builder.py` script will take the development environment and setup an (almost)ready-to-use package for the respective platform.

1. Optionally, Run InnoInstaller script (applies only to Windows releases)

Interactive Debugger:

To use, make sure MFB is running with a visible console Press the ~ key when mfb is running. MFB should freeze, and you should see this in the console:

```
Python 3.3.3 (default, Nov 25 2013, 20:20:09)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>>
```

Use this interactive console to inspect any variable or issue arbitrary commands. For example:

```
>>>> logic.mvb.objects
```

will print out the list of objects in the scene

You can also modify the state of the game.

```
>>> logic.logger.new("hello there")
```

will print out “hello there” in the top part of the screen once the game resumes.

Press `Ctrl+D` in the console will quit the python interpreter and send you back to the game.

Model

All the scene data for an active flipbook session is stored in the singleton object MVBScene. The class is accessible from logic.mvb

Two main data containers are:

- logic.mvb.objects{} for all data related to the objects in the scene.
- logic.mvb.slides[] for all data related to the animation timeline.

Although in many cases, it is better to modify these data through other helper functions, rather than directly. i.e. logic.timeline.slideAdd() and logic.timeline.slideDelete() to manipuate slides.

File saving and loading is done mostly by converting the above class to and from json file.

View

The view is a 3D scene powered by the blenderplayer with a 2D GUI controls overlayed on top. There are two handlers for mouse and keyboard inputs:

- By `Framework` for Blender 3D scene (i.e. mouse over 3D object, session shortcuts);
- By `BGUI` for 2D GUI layer (i.e. mouse over buttons and text input)

Program Data:

When running, MFB will create a folder under `~/Flipbook/` as its temporary folder. Error log, data cache, and program data will be stored to this folder. This folder is safe to delete.