Forum: Moteur de jeu GameBlender et alternatives

Topic: Runtime error ...

Subject: Re: Runtime error ...

PostÃ© par: tales

Contribution le : 17/1/2012 6:00:08

J'ai eu le même soucis essaie de remplacer le script game_engine_save_as_runtime par

```
# ##### BEGIN GPL LICENSE BLOCK #####
#
#  This program is free software; you can redistribute it and/or
#  modify it under the terms of the GNU General Public License
#  as published by the Free Software Foundation; either version 2
#  of the License, or (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software Foundation,
#  Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
#
# ##### END GPL LICENSE BLOCK #####

bl_info = {
   &#039;name&#039;: &#039;Save As Game Engine Runtime&#039;,
   &#039;author&#039;: &#039;Mitchell Stokes (Moguri)&#039;,
   &#039;version&#039;: (0, 3, 1),
   "blender": (2, 6, 1),
   "api": 42107,
   &#039;location&#039;: &#039;File > Export&#039;,
   &#039;description&#039;: &#039;Bundle a .blend file with the Blenderplayer&#039;,
   &#039;warning&#039;: &#039;&#039;,
   &#039;wiki_url&#039;: &#039;http://wiki.blender.org/index.php/Extensions:2.5/Py/&#039;\
      &#039;Scripts/Game_Engine/Save_As_Runtime&#039;,
   &#039;tracker_url&#039;: &#039;https://projects.blender.org/tracker/index.php?&#039;\
      &#039;func=detail&aid=23564&#039;,
   &#039;category&#039;: &#039;Game Engine&#039;}

import bpy
import os
import sys
import shutil
import tempfile
```

```
def CopyPythonLibs(dst, overwrite_lib, report=print):
    import platform

    # use python module to find pytohn&#039;s libpath
    src = os.path.dirname(platform.__file__)

    # dst points to lib/, but src points to current python&#039;s library path, eg:
    #  &#039;/usr/lib/python3.2&#039; vs &#039;/usr/lib&#039;
    # append python&#039;s library dir name to destination, so only python&#039;s
    # libraries would be copied
    if os.name == &#039;posix&#039;:
        dst = os.path.join(dst, os.path.basename(src))

    if os.path.exists(src):
        write = False
        if os.path.exists(dst):
            if overwrite_lib:
                shutil.rmtree(dst)
                write = True
        else:
            write = True
        if write:
            shutil.copytree(src, dst, ignore=lambda dir, contents: [i for i in contents if i ==
&#039;__pycache__&#039;])
    else:
        report({&#039;WARNING&#039;}, "Python not found in %r, skipping pythn copy" % src)


def WriteAppleRuntime(player_path, output_path, copy_python, overwrite_lib):
    # Enforce the extension
    if not output_path.endswith(&#039;.app&#039;):
        output_path += &#039;.app&#039;

    # Use the system&#039;s cp command to preserve some meta-data
    os.system(&#039;cp -R "%s" "%s"&#039; % (player_path, output_path))

    bpy.ops.wm.save_as_mainfile(filepath=os.path.join(output_path,
"Contents/Resources/game.blend"),
                                relative_remap=False,
                                compress=False,
                                copy=True,
                                )

    # Python doesn&#039;t need to be copied for OS X since it&#039;s already inside
blenderplayer.app


def WriteRuntime(player_path, output_path, copy_python, overwrite_lib, copy_dlls, report=print):
    import struct
```

```
    # Check the paths
    if not os.path.isfile(player_path) and not(os.path.exists(player_path) and
player_path.endswith(&#039;.app&#039;)):
        report({&#039;ERROR&#039;}, "The player could not be found! Runtime not saved")
        return

    # Check if we&#039;re bundling a .app
    if player_path.endswith(&#039;.app&#039;):
        WriteAppleRuntime(player_path, output_path, copy_python, overwrite_lib)
        return

    # Enforce "exe" extension on Windows
    if player_path.endswith(&#039;.exe&#039;) and not output_path.endswith(&#039;.exe&#039;):
        output_path += &#039;.exe&#039;

    # Get the player&#039;s binary and the offset for the blend
    file = open(player_path, &#039;rb&#039;)
    player_d = file.read()
    offset = file.tell()
    file.close()

    # Create a tmp blend file (Blenderplayer doesn&#039;t like compressed blends)
    tempdir = tempfile.mkdtemp()
    blend_path = os.path.join(tempdir, bpy.path.clean_name(output_path))
    bpy.ops.wm.save_as_mainfile(filepath=blend_path,
                                relative_remap=False,
                                compress=False,
                                copy=True,
                                )
    blend_path += &#039;.blend&#039;

    # Get the blend data
    blend_file = open(blend_path, &#039;rb&#039;)
    blend_d = blend_file.read()
    blend_file.close()

    # Get rid of the tmp blend, we&#039;re done with it
    os.remove(blend_path)
    os.rmdir(tempdir)

    # Create a new file for the bundled runtime
    output = open(output_path, &#039;wb&#039;)

    # Write the player and blend data to the new runtime
    print("Writing runtime...", end=" ")
    output.write(player_d)
    output.write(blend_d)

    # Store the offset (an int is 4 bytes, so we split it up into 4 bytes and save it)
```

```
    output.write(struct.pack(&#039;B&#039;, (offset>>24)&0xFF))
    output.write(struct.pack(&#039;B&#039;, (offset>>16)&0xFF))
    output.write(struct.pack(&#039;B&#039;, (offset>>8)&0xFF))
    output.write(struct.pack(&#039;B&#039;, (offset>>0)&0xFF))

    # Stuff for the runtime
    output.write(b&#039;BRUNTIME&#039;)
    output.close()

    print("done")

    # Make the runtime executable on Linux
    if os.name == &#039;posix&#039;:
        os.chmod(output_path, 0o755)

    # Copy bundled Python
    blender_dir = os.path.dirname(bpy.app.binary_path)
    runtime_dir = os.path.dirname(output_path)

    if copy_python:
        print("Copying Python files...", end=" ")
        py_folder = os.path.join(bpy.app.version_string.split()[0], "python", "lib")
        dst = os.path.join(runtime_dir, py_folder)
        CopyPythonLibs(dst, overwrite_lib, report)
        print("done")

    # And DLLs
    if copy_dlls:
        print("Copying DLLs...", end=" ")
        for file in [i for i in os.listdir(blender_dir) if i.lower().endswith(&#039;.dll&#039;)]:
            src = os.path.join(blender_dir, file)
            dst = os.path.join(runtime_dir, file)
            shutil.copy2(src, dst)

        print("done")

from bpy.props import *


class SaveAsRuntime(bpy.types.Operator):
    bl_idname = "wm.save_as_runtime"
    bl_label = "Save As Game Engine Runtime"
    bl_options = {&#039;REGISTER&#039;}

    if sys.platform == &#039;darwin&#039;:
        # XXX, this line looks suspicious, could be done better?
        blender_bin_dir = &#039;/&#039; +
os.path.join(*bpy.app.binary_path.split(&#039;/&#039;)[0:-4])
        ext = &#039;.app&#039;
```

```
else:
    blender_bin_path = bpy.app.binary_path
    blender_bin_dir = os.path.dirname(blender_bin_path)
    ext = os.path.splitext(blender_bin_path)[-1].lower()

default_player_path = os.path.join(blender_bin_dir, &#039;blenderplayer&#039; + ext)
player_path = StringProperty(
        name="Player Path",
        description="The path to the player to use",
        default=default_player_path,
        subtype=&#039;FILE_PATH&#039;,
        )
filepath = StringProperty(
        subtype=&#039;FILE_PATH&#039;,
        )
copy_python = BoolProperty(
        name="Copy Python",
        description="Copy bundle Python with the runtime",
        default=True,
        )
overwrite_lib = BoolProperty(
        name="Overwrite &#039;lib&#039; folder",
        description="Overwrites the lib folder (if one exists) with the bundled Python lib folder",
        default=False,
        )

# Only Windows has dlls to copy
if ext == &#039;.exe&#039;:
    copy_dlls = BoolProperty(
            name="Copy DLLs",
            description="Copy all needed DLLs with the runtime",
            default=True,
            )
else:
    copy_dlls = False

def execute(self, context):
    import time
    start_time = time.clock()
    print("Saving runtime to %r" % self.filepath)
    WriteRuntime(self.player_path,
                 self.filepath,
                 self.copy_python,
                 self.overwrite_lib,
                 self.copy_dlls,
                 self.report,
                 )
    print("Finished in %.4fs" % (time.clock()-start_time))
    return {&#039;FINISHED&#039;}
```

```
    def invoke(self, context, event):
        if not self.filepath:
            ext = &#039;.app&#039; if sys.platform == &#039;darwin&#039; else
os.path.splitext(bpy.app.binary_path)[-1]
            self.filepath = bpy.path.ensure_ext(bpy.data.filepath, ext)

        wm = context.window_manager
        wm.fileselect_add(self)
        return {&#039;RUNNING_MODAL&#039;}


def menu_func(self, context):
    self.layout.operator(SaveAsRuntime.bl_idname)


def register():
    bpy.utils.register_module(__name__)

    bpy.types.INFO_MT_file_export.append(menu_func)


def unregister():
    bpy.utils.unregister_module(__name__)

    bpy.types.INFO_MT_file_export.remove(menu_func)


if __name__ == "__main__":
    register()
```