# Game Teleporter: A Development Tool For Everyone

Tony Morelli and Dwight Egbert
Computer Science and Engineering Department University of Nevada Reno
morelli@cse.unr.edu, egbert@cse.unr.edu
Reno, Nevada 89577

*Abstract*

**The Game Teleporter is an application that translates one file format to another without any user intervention. The application itself is made up of different combinations of input plug-ins and output plug-ins. For example, the application can take an Adobe Flash file as an input and then output a Playstation Portable binary that can be run on the specified target platform and have the same functionality as the original Adobe Flash file. Flash is a file format commonly used by web developers to create animated and interactive websites. Using Flash as an input opens up development to people who might have great ideas for games, but cannot display them on the desired target as they do not know how to write programs for it. This tool is applicable at all levels of education. A very simple input/output plug-in combination allows elementary school children to create a game simply by drawing pictures. This can enhance their interest in learning computer programming. As shown in this paper, students used this tool with a customized graphical input plug-in to create a basic game that will run on both Microsoft Windows and on a Sony Playstation Portable. College students can also utilize this application by creating and modifying the plug-ins to adapt to whatever the current technology requires. This paper lays out how the Game Teleporter functions, and how it can be used in education at all levels to generate better computer programmers for the future.**

*Index Terms* – Game Development, Multiple Platforms, Open Source, Playstation Portable.

## INTRODUCTION

Having students starting to get interested in software at the earliest of ages benefits the entire software industry. The Game Teleporter tool encourages young students to begin thinking about writing software as a potential career at a young age. The input/output plug-in architecture allows for simple input files to create complex output applications. As shown in this document, a simple drawing can be used to create an interactive application running on a target platform of interest to young students. In this example, a drawing created in Adobe Photoshop is used as an input, and as an output it is displayed on a Playstation Portable. The user needs to do nothing besides draw the picture, and click a button on the user interface. All the code generation is done without any user interaction. Getting an application up and running on an entertainment device used by the child should promote an interest in the software field.

Also, this tool can be used by advanced students to develop plug-ins that revolve around the newest of technologies. A developer can create or modify plug-ins in order to generate complex applications on multiple target platforms with a simple input format that can be generated by everyone. Input plug-ins can also be made to decipher complex file formats such as a binary file from a specific target platform.

This project will get the attention of a young perspective programmer and allow that person to stay with this project all the way up to becoming an advanced developer. This tool will allow developers to grow with technologies and skills while working within the same product that is very familiar to the budding programmer. This paper shows a specific example of one input plug-in and one output plug-in, however the tool itself was crafted in such a way that adding or modifying plug-ins can be easily done. The application was created using Microsoft's Visual Studio .net in the C++ programming language, and built to run on the Microsoft Windows XP operating system, however the code is intended to be portable in such a way that the application can be built and run on any platform with a C++ compiler. All components of the Game Teleporter are open source and available on the Game Teleporter website. The open source nature will promote learning be examining the source code, and improve the overall quality of the product by making it available for modification by anyone.

## SIMILAR SOFTWARE

This product is comparable to a few different products. The first product is a code converter. There are several available on the web, and they work to varying degrees of success. Most of them convert C# to VB.Net, which is a fairly straight forward process as the languages are very similar. Most of these converters did a pretty poor job of converting, and I never was able to get one translated piece of code to actually

generate an application of similar functionality. Also, all these converters were basically converting to the same platform. They were not creating binaries for different targets. So in that respect, the Game Teleporter works better than the code converters and has more practical applications.

The next product this is similar to is a game development kit for beginners. One in particular is entitled '*The Game Makers Apprentice'*. This book and companion development environment CD are geared towards someone just learning the basics of game design. Most of the book deals with sprites and assigning events and actions to the sprites. This is a really good way to teach concepts to budding young programmers. Near the end of the book the author introduces the concept of GML, the Game Maker Language. This is a basic scripting language that the author uses to teach such programming basics as FOR loops and IF statements. Once the game is created it can only be run on a windows PC. It does teach the necessary elements of programming, but it also misses out on catching a child's interest by showing a game running on a different target (such as a PSP). I think this product makes a great companion to the Game Teleporter. An input plugin could be created to take the Game Maker source, and convert it to a game that could be run on a PSP with the simple click of one button. This will also make use of the nice environment of the Game Maker. The simple input plugin for the pong game used for this paper will only be of interest to a certain type of person, where as using the Game Maker will help more with getting a beginner introduced to basic programming concepts.

Another program, *Game Editor,* allows the exporting to a few different platforms such as the PocketPC, however the overall quality of the product is not as good as Game Maker and you are limited to using the Game Editor environment to create the game. The *Game Teleporter* has the best of all worlds. With the plug-in architecture, and the open source nature of the product any and all plug-in combinations are possible. You will be able to use the development environment of your choice and play the game on the target of your choice. You will also have all the converted source code for each target platform specified which will make learning about that target much easier.

## STUDY

A study was conducted to determine how useful this product is for educational purposes. The study consisted of 18 individuals ranging in ages from 13-50, and skills ranging from no computer programming interest, to computer gamer, to seasoned software engineer. The study began with me explaining what the product does, followed by showing the participant the final product they were about to create, which ended up being a simple Pong type of a game. I then ran the input plug-in, which in this case was an interface allowing the user to draw their own characters that would be used in the game. The characters were (1) Your Paddle, (2) Opponent Paddle, and (3) the ball. The software was run, and the student drew the desired components on a tablet PC using the stylus as a method of drawing. The student simply had to draw on the screen as if holding a pen. Once the drawings had been completed, the Game Teleporter generated all source code to create a Visual Basic Program and a Playstation Portable game, then the Teleporter built the programs for each target. I then started up the Visual Basic application and let the student play the game. Next, I started up the application on the Playstation Portable and again let the student play the game. I then asked the students a series of questions about what had just taken place.

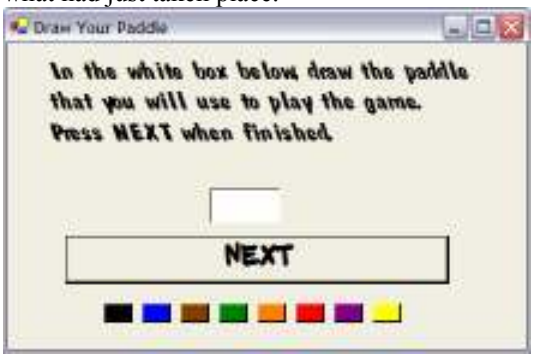


FIGURE 1 USER INTERFACE

(1) Did they think it would be that easy to create a game?
(2) Was the game interesting?
(3) Which game did they like better, the Psp or the Visual Basic?
(4) Did using this application make them want to be a computer programmer?
(5) Did they have any questions?

The results were mixed. I was so excited about this product, that I assumed everyone would immediately want to become a computer programmer. That was not the response I received. Everyone did not think it would be that easy to make a game run on a Psp. The non programmers were also impressed with the ability to make a PC game that quickly.

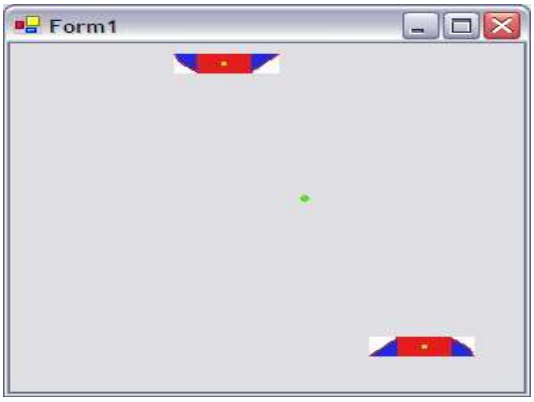

FIGURE 2: SAMPLE VISUAL BASIC.NET GAME

The most common response to 'Was the game interesting?' was that it would be more interesting if the game kept score. Younger gamers wanted to know how to beat the game. The basic game had no scoring mechanism, just simply pong. If you wanted to keep score you would have to use a pencil and paper. Seasoned engineers did not care whether or not the game was interesting as they were more interested in how the process took place.

The survey of the Psp version or the Visual Basic Version was split evenly amongst the age groups. The older group (college age and above) were much more interested in the Visual Basic program as they could use it on their own PC. The younger group was more interested in the Psp version. This was due to the 'cool factor' of Sony's portable. Although none of the kids in the survey owned a Psp, they all knew someone who did, and the kids in the survey always spoke of the Psp owner as the 'cool kid'. I think the children were more interested in the Psp version because if they could make a game that could run on the cool kid's toy, they too would be cool.

When I asked if using this application made them want to become a computer programmer, I received mixed results. This question was only asked to the non programmers in the study. I could have predicted the answer to this question within the first 2 minutes of starting the study with each individual. Either the person was already interested in programming or not. At first I considered this a failure on my part as the goal of this was to get kids interested in programming.

The initial failure of the previous question proved to be incorrect as I interviewed more and more people about the product. The longest part of the survey was always the last question. The engineers wanted to know in detail how everything worked. How the intermediate code was generated, how the code was compiled, and how did I get my own application to run on the Psp. The non engineers who wanted to be programmers would ask questions about how to write programs, and I would walk them through some of the code showing them how each line could make a difference in the final product. If the student was interested in learning the basics of computer programming, I also showed them the QBasic program that was also output. This has all the source code in one file and made it easier for the student to see how the program flowed. It also pointed out the limitations of the basic language as QBasic only supports lower video resolutions which made the graphics look blocky. Overall, however, the QBasic source code served its purpose as it is much easier to understand than the Psp source code that was generated. Some kids were interested in programming but did not have the most basic computer skills, so their questions revolved around moving files from the computer to the Psp. Some asked why the same file could not be run on both the PC and on the Psp.

The group I feared the most when I asked the last question was the non-programmers of the group. I figured since this product did not make them want to be programmers, I would be met with blank stares. Fortunately, that was not the case. This group had the most suggestions about how to make things better. They suggested a better drawing interface with the ability to move in pictures from outside sources. They wanted a scoring system in the game so it would be more fun and more challenging to play. They wanted to add sounds. One child made the comment that if everyone was drawing their own character the quality of the game would be diminished because not everyone could draw. The overall game would be better if each person playing the game could create content that they specialized in.

The comments from the last question were the best for this product. Even though this product did not make programmers out of everyone, it made everyone think about how to make things better. And more importantly what role each individual could have in making things better. Not everyone is interested in programming, and not everyone is interested in drawing, however everyone surveyed found some talent of their own that they wanted to use to make this product better. That is exactly how a development team works. Making a game consists of story writers, programmers, graphic artists, audio creators and more. At the very least this product makes anyone think about what role they could play in a future game development team.

## METHODOLOGY

The Game Teleporter is made up of 3 basic modules, Input Plug-ins, Output plug-ins, and an intermediate state. The basic flow of the software is as follows. In stage 1 the input plug-in creates an intermediate set of files. In stage 2, the output plug-in uses the intermediate set of files to create the desired output. Having a standard intermediate file set format makes it possible to convert from any one file type to any other file type. All input plug-ins must have the capability of going from the input file, or set of files, to the known intermediate state. The output plug-ins all must support starting from a common intermediate set of files and translate them to the desired output. The following sections describe the process the application takes to convert a Photoshop (Psd) file into a Playstation Portable binary object that can be run on the Playstation Portable gaming system. Although these descriptions are specific to the Psd input plug-in and the Psp output plug-in, the methodologies used can be applied to create new input and/or output plug-ins.

### *Input Plug-ins*

As described earlier, input plug-ins must support converting any files, or set of files into an intermediate set of files. These files are grouped into two areas, source code and resources. The source code contains commands in a simple language similar to the C programming language. Currently, the only resources supported are images stored in the Portable Network Graphics (Png) format. Other resources that can be added at a later date include sounds, 3 Dimensional Graphics, and other data files.

The Photoshop input plug-in is designed to create a slide show of images that are contained within a Photoshop file. Adobe Photoshop has support for multiple layers. A layer is similar to a page in a notebook made up of transparent paper. A drawing exists on each page of the note book, and when laid on top of each other it is possible to see all images at once. That is what is seen when the Photoshop document is first opened. The Photoshop Plug-in's function is to take the different layers (pages in the notebook), break them out into separate images, and then generate code to display them one at a time keeping the location of the images the same as where they were in the original Photoshop file.

To achieve this goal, an open source tool was utilized. *Imagemagick* is an open source and freely distributed graphics package which contains several utilities to manipulate graphics. Although it is very possible to write complete software solutions not using open source products, they are very useful and it avoids re-inventing the wheel for known problems. In the first stage of the Psd conversion process, the Imagemagick utility *convert* is ran on the Psd file. When convert is run on a Psd file, the output of the command is one Png file for every layer contained within the file. This is a very important step to have automatically taken care of by an outside utility, however the convert utility is missing two important pieces of information. It is missing the location of the image within the layer, and it is also missing the name of the layer. The position within the file is needed to properly display the image, and the layer name is needed for potential key word usage later if the Psd plug-in is desired to do more than a simple slide show.

To obtain the two missing pieces of data, the position of each layer, and the layer name, the Game Teleporter was written to get these pieces directly out of the Psd file itself. The Psd plug-in is a stripped down version of the Adobe Photoshop 6.0 File Formats Specification. It is only concerned with the name of each layer and the location of each layer. Nothing more. The plug-in will display all sorts of information about the file and about each layer, however that information is simply displayed and not saved for later use. The plug-in can be modified in such a way that available and not currently stored data can be stored, but that is not necessary for the purpose of this plug-in.**** The code was written to follow the spec and upon completion of processing all data within the Psd file, the plug-in is aware of layer names, and the corresponding bounds which include the coordinates of the Top Left corner, and the Bottom Right corner of the graphic contained on each layer.****

Knowing the layer names, the plug-in then makes modifications to the Png files created by the convert program earlier. The convert program knows nothing about layer names, so it names the files according to the layer number. The output is a number followed by .png (i.e. 1.png, 2.png, 3.png, etc…). This is somewhat not useful, but fortunately the layers are processed in the same order by the Game Teleporter. So the Psd plug-in can simply rename each file based on an index into an array of filenames generate by the psd plug-in when it read the file earlier. At this point, the input plug-in has created the resources necessary for the intermediate stage and copies them to an intermediate directory. The final step of the input plug-in stage is to generate the source code needed for an output plug-in to create the desired output.

The Psd input plug-in creates two source code files. The first file created is named Pictures.h and contains a known structure for pictures. The information stored within that file contains the path name of the image resources, and their associated bounds. A sample Pictures.h file is shown below:

```
/*****************************/
static struct Picture picts[] = {
  {"Green_Box.png",1,1,34,45},
  {"Red_Box.png",0,432,35,479},
  {"Blue_Box.png",224,0,265,51},
  {"Yellow_Box.png",226,427,270,478}
          };
#define TOTAL_PICTURES 4
/*****************************/
```

FIGURE 3: PICTURES.H

The above code is generated at runtime by the Psd plug-in and includes all relevant information including the total number of pictures.

The second piece of code generated by the Psd Input Plug-in is the code that describes the necessity for displaying the slide show. This code is formatted in a C like syntax in order for ease of conversion for most out put plug-ins. Most platforms support some kind of C compiler, so that is why that format was chosen. The main code piece not only contains commands, but it also contains XML markers designating different sections of the code. The xml tags generated by the Psd Plug-in include, Includes, Declarations, and Mainloop. The includes section is designed to describe any external files created by the plug-in, which in this case is Pictures.h. The Declarations section is much like a declaration section in a C program where it outlines what variables are about to be used. The mainloop is a section which contains the actual commands to generate the desired output. A sample output of the Psd Plug-in code generation is shown below:

```
/*****************************/
```

```
<INCLUDES>
#include "Pictures.h"
</INCLUDES>
<DECLARATIONS>
int picIndex = 0;
InputButton button;
</DECLARATIONS>
<MAINLOOP>
  GetInput();
  if (button & RIGHT)
  {
    ClearScreen();
    picIndex++;
    if (picIndex >= TOTAL_PICTURES)
    {
      picIndex = 0;
    }
    DisplayImage(picts[picIndex]);
  }
  if (button & LEFT)
  {
    ClearScreen();
    picIndex--;
    if (picIndex < 0 )
    {
      picIndex = TOTAL_PICTURES-1;
    }
    DisplayImage(picts[picIndex]);
  }
</MAINLOOP>
/*****************************/
```

FIGURE 4: INTERMEDIATE CODE SAMPLE

A quick run through of the code shows that our main loop calls a function GetInput which is then assumed to place whatever buttons are pressed in a variable labeled button. Then based on what button is pressed (either right or left), the screen is cleared and the next image is displayed. This keeps going on forever. This is a pretty simple state machine, and it will be shown later how this simple code is turned into Playstation Portable Code.

### *Output Plug-ins*

Output plug-ins are designed to utilize the code and resources generated by the input plug-in to create a desired result. The plug-in discussed here is a Playstation Portable plug-in, but the concepts discussed here can be used to create plug-ins for different platforms.

The Psp Output Plug-in begins with a skeleton set of files which act as a starting point for a Psp Project. These are grouped into two categories, Necessary Files, and Modifiable files. The Necessary Files are files necessary to build a Psp Executable, and are not able to be modified by the Psp plug-in. These files are simply copied into every project directory. The other set of files, the Modifiable Files, are files necessary to build a Psp Binary, however these need to be modified based on the output of the input plug-in.

The two files being modified are *Makefile*, and *main.c*. The changes to Makefile are extremely basic. The Makefile contains two identifiers for each Psp Project, Target and Title. The Target defines the executable name and shall not contain any spaces. The Title is what appears as the title on the Playstation Portable Game Selection Screen and may contain spaces. Both of these configuration options are entered on the command line when the application is started and inserted into the file when the Psp Output Plug-in is run.

The second and most complicated modifiable file in the Psp output plug-in is main.c. The skeleton main.c file contains all the include files, functions, and definitions required for all Psp applications. It also includes keywords such that the Psp Output Plug-in can insert the necessary code at the correct places within the file. The file contains XML tags for Includes, Declarations, and the MainLoop. Since this plug-in is creating an application to be built by a C compiler, the plug-in can simply copy in all includes defined in the intermediate code file. They will be a direct drop in. The same goes for the declarations with one exception. The variables are declared in the same fashion as in the intermediate code, however the Psp Output Plug-in must maintain an internal list of variables for use which will be described later in this document. The variable types supported by this plug-in are an Integer and a Button type.

After the declarations, the MainLoop tag is encountered and the guts of the code are added from the intermediate code generated by the input plug-in. The Psp plug-in reads through the intermediate code and makes modifications as necessary. For example, any time the intermediate code makes a call to GetInput(), the Psp plug-in makes a call to ProcessInputs, and stores the results into its variable that has been declared as a button. In this case, the variable name is *button.* Another substitution deals with the use of the variable. Whenever the intermediate code checks the value of *button*, the Psp output plug-in generates code looking at the value of *button.buttons* as that is the naming convention for buttons in the Psp world. The intermediate code generated by the input plug-in as shown above is translated into the following code which is understood by the Playstation Portable compiler.

```
/*************************/
  button = ProcessInputs();
  if (button.Buttons & RIGHT)
  {
    ClearCurrentScreen();
    picIndex++;
    if (picIndex >= TOTAL_PICTURES)
    {
      picIndex = 0;
```

```
  }
  DisplayImage(picts[picIndex]);
 }
 if (button.Buttons & LEFT)
 {
  ClearCurrentScreen();
  picIndex--;
  if (picIndex < 0 )
  {
   picIndex = TOTAL_PICTURES-1;
  }
  DisplayImage(picts[picIndex]);
 }
/*************************/
```

FIGURE 5: PSP OUTPUT PLUG-IN GENERATE CODE

As shown above the code translates pretty easily into Psp native code, but also illustrates why a C compiler cannot be relied upon to do all conversions.

With all the code in place, the plug-in then copies all resources into the destination directory. This sets the game up to be built. One last modification needs to be made prior to building the game. A build script is generated which contains the name and directory of where the source files will be placed. A batch file is then run which will enter the build environment and build the Playstation Portable Binary image.

When the newly created application is run on the actual hardware, a slideshow is presented to the user. The slideshow cycles through all the layers originally drawn within the Photoshop file and displays them in their original location. This simple example illustrates the power of this tool. From a Psd file, to running on real hardware in less than 1 minute is something not shy of amazing.

This software was designed to be run completely from the command line as it will enable people to use the utility in scripts that will automatically generate a desired output. This, however, is not the best approach for people who are using this tool as a development kit for non programmers. These people may be intimidated by the requirement of using a command prompt. A front end for the command prompt, *GT Front End*, was created. This utility allows the user to input all required command line options, such as the input and output files, and the output file names into a Graphical User Interface (GUI) which is in a windowed environment that the user is very comfortable using. This will promote the use of this tool by people who are not completely computer savvy.

## CONCLUSION

This paper has demonstrated how the application can translate one file format into another. It shows how easy new plug-ins can be created for either a new input or a new output format. The application was designed with the intention of it to be used by everyone at all programming skill levels. A younger student who desires to create a game can do so by simply drawing pictures and running it through the proper plug-ins. A higher level student can write more plug-ins, or modify the existing ones to demonstrate programming abilities, or to learn new ones. This broad range of users and uses makes this tool a great part of the education process. The ease of adding new plug-ins will keep students very interested, as well as allowing this piece of software to evolve with time.

***References:***

[1] Game Teleporter Website. http://www.tonymorelli.com/gtp/

[2] Adobe Photoshop 6.0 File Formats Specification Version 6.0 Release 2 November 2000. Copyright 1991-2000 Adobe Systems Incorporated

[3] Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8 Copyright 2005 Adobe Systems Incorporated

[4] ImageMagick , http://www.imagemagick.org

[5] Greg Perry (1993) *QBASIC By Example.* Que Publishing

[6] Habgood and Overmars (2006) *Game Maker*. APress

[7] Watrall and Herber (2004) *Flash MX 2004.* Savvy