O Programa Weaver

Thiago Leucz Astrizi

thiago@bitbitbit.com.br

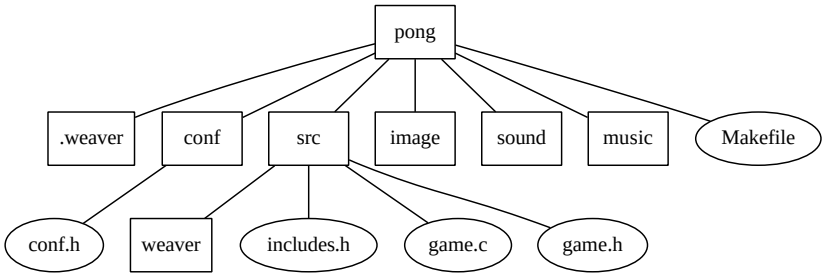
Abstract: *This article describes using literary programming the program Weaver. This program is a project manager for the Weaver Game Engine. If a user wants to create a new game with the Weaver Game Engine, they use this program to create the directory structure for a new game project. They also use this program to add new source files and shader files to a game project. And to update a project with a more recent Weaver version installed in the computer. The presenting code in C is cross-platform and should work under Windows, Linux, OpenBSD and possibly other Unix variants.*

1. Introduction

A game engine is made by a set of libraries and functions that helps a game creation offering common functionalities for this kind of development. But besides the libraries and functions, there should exist a manager responsible for creating some code which uses the library in a correct way and executes the necessary initializations.

The Weaver Game Engine has very strict prerequisites about how the directory with a game project should be organized. To follow these requisites, this program is necessary. It initializesin a correct way the directory structure in a new project. It adds new source files with the correct code to ensure the code integration. And controlling the project in this way, it also knows how to perform updates in the libraries for more recent versions.

This program usage is by the command line. For example, if a user types "`weaver pong`", a new directory structure like in the following image will be created.



As seguintes sees do artigo esto organizadas da seguinte forma. A seo 2 abordar a licensa do software. A seo 3 listar as variveis usadas para controlar seu comportamento. A seo 4 trar algumas macros que usaremos, algumas das quais apareceram na estrutura do programa. A seo 5 apresentar algumas funes auxiliares que utilizaremos. A seo 6 mostrar a inicializao das variveis do programa. A seo 7 mostrar os casos de uso do programa e como implement-los aps termos as variveis com os valores certos.

2. Copyright e licenciamento

Segue abaixo a licena do programa e sua traduo no-oficial:

A verso completa da licena pode ser obtida junto ao cdigo-fonte Weaver ou consultada no link mencionado.

3. Variveis e Estrutura do Programa Weaver

O comportamento de Weaver deve depender das seguintes variveis:

`inside_weaver_directory` : Indicar se o programa est sendo invocado de dentro de um projeto Weaver.

`argument` : O primeiro argumento, ou NULL se ele no existir

`argument2` : O segundo argumento, ou NULL se no existir.

`project_version_major` : Se estamos em um projeto Weaver, qual o maior nmero da verso do Weaver usada para gerar o projeto. Exemplo: se a verso for 0.5, o nmero maior 0. Em verses de teste, o valor sempre 0.

`project_version_minor` : Se estamos em um projeto Weaver, o valor do menor nmero da verso do Weaver usada para gerar o projeto. Exemplo, se a verso for 0.5, o nmero menor 5. Em verses de teste o valor sempre 0.

`weaver_version_major` : O nmero maior da verso do Weaver sendo usada no momento.

`weaver_version_minor` : O nmero menor da verso do Weaver sendo usada no momento.

`arg_is_path` : Se o primeiro argumento ou no um caminho absoluto ou relativo para um projeto Weaver.

`arg_is_valid_project` : Se o argumento passado seria vlido como nome de projeto Weaver.

`arg_is_valid_module` : Se o argumento passado seria vlido como um novo mdulo no projeto Weaver atual.

`arg_is_valid_plugin` : Se o segundo argumento existe e se ele um nome vlido para um novo plugin.

`arg_is_valid_function` : Se o segundo argumento existe e se ele seria um nome vlido para um loop principal e tambm para um arquivo.

`project_path` : Se estamos dentro de um diretrio de projeto Weaver, qual o caminho para a sua base (onde h o Makefile)

`have_arg` : Se o programa invocado com argumento.

`shared_dir` : Dever armazenar o caminho para o diretrio onde esto os arquivos compartilhados da instalao de Weaver. Por padro, ser igual "`/usr/local/share/weaver`", mas caso exista a varivel de ambiente `WEAVER_DIR`, ento este ser considerado o endereo dos arquivos compartilhados.

`author_name` , `project_name` e `year` : Contero respectivamente o nome do usurio que est invocando Weaver, o nome do projeto atual (se estivermos no diretrio de um) e o ano atual. Isso ser importante para gerar as mensagens de Copyright em novos projetos Weaver.

`return_value` : Que valor o programa deve retornar caso o programa seja interrompido no momento atual.

A estrutura geral do programa com a declarao de todas as variveis ser:

Arquivo: `src/weaver.c`:

<Seo a ser Inserida: Cabealhos Includos no Programa Weaver>
<Seo a ser Inserida: Macros do Programa Weaver>
<Seo a ser Inserida: Funes auxiliares Weaver>

```
int main(int argc, char **argv){
  int return_value = 0; /* Valor de retorno. */
  bool inside_weaver_directory = false, arg_is_path = false,
    arg_is_valid_project = false, arg_is_valid_module = false,
    have_arg = false, arg_is_valid_plugin = false,
    arg_is_valid_function = false; /* Variveis booleanas. */
  unsigned int project_version_major = 0, project_version_minor = 0,
    weaver_version_major = 0, weaver_version_minor = 0,
    year = 0;
  /* Strings UTF-8: */
```

```
  char *argument = NULL, *project_path = NULL, *shared_dir = NULL,
    *author_name = NULL, *project_name = NULL, *argument2 = NULL;
                                <Seo a ser Inserida: Inicializao>
            <Seo a ser Inserida: Caso de uso 1: Imprimir ajuda (criar projeto)>
          <Seo a ser Inserida: Caso de uso 2: Imprimir ajuda de gerenciamento>
                      <Seo a ser Inserida: Caso de uso 3: Mostrar verso>
                <Seo a ser Inserida: Caso de uso 4: Atualizar projeto Weaver>
                    <Seo a ser Inserida: Caso de uso 5: Criar novo mdulo>
                   <Seo a ser Inserida: Caso de uso 6: Criar novo projeto>
                    <Seo a ser Inserida: Caso de uso 7: Criar novo plugin>
                    <Seo a ser Inserida: Caso de uso 8: Criar novo shader>
                <Seo a ser Inserida: Caso de uso 9: Criar novo loop principal>
END_OF_PROGRAM:
                                <Seo a ser Inserida: Finalizao>
  return return_value;
}
```

4. Macros e Cabealhos do Programa Weaver

O programa precisar de algumas macros. A primeira delas dever conter uma string com a verso do programa. A verso pode ser formada s por letras (no caso de verses de teste) ou por um nmero seguido de um ponto e de outro nmero (sem espaos) no caso de uma verso final do programa.

Para a segunda macro, observe que na estrutura geral do programa vista acima existe um rtulo chamado `END_OF_PROGRAM` logo na parte de finalizao. Uma das formas de chegarmos l por meio da execuo normal do programa, caso nada d errado. Entretanto, no caso de um erro, ns podemos tambm chegar l por meio de um desvio incondicional aps imprimirmos a mensagem de erro e ajustarmos o valor de retorno do programa. A responsabilidade de fazer isso ser da segunda macro.

Por outro lado, podemos tambm querer encerrar o programa previamente, mas sem que tenha havido um erro. A responsabilidade disso da terceira macro que definimos.

Seo: Macros do Programa Weaver:

```
#define VERSION "Alpha"
#define W_ERROR() {perror(NULL); return_value = 1; goto END_OF_PROGRAM;}
#define END() goto END_OF_PROGRAM;
```

Como estamos usando a funo de biblioteca `perror`, devemos incluir o cabealho `stdio.h`, o que tambm nos trar s funes de imprimir na tela, abrir e fechar arquivos e escrever neles, o que nos ser til. Vamos inserir suporte valores booleanos que usamos na prpria estrutura do programa e tambm a biblioteca padro, que tem funes como `exit` usadas na estrutura do programa:

Seo: Cabealhos Includos no Programa Weaver:

```
#include <stdio.h> // printf, fprintf, fopen, fclose, fgets, fgetc, perror
#include <stdbool.h> // bool, true, false
#include <stdlib.h> // free, exit, getenv
```

5. Funes Auxiliares

Listemos aqui algumas funes que usaremos ao longo do programa para facilitar sua descrio.

5.1. path_up: Manipula Caminho

Para manipularmos o caminho da rvore de diretrios, usaremos uma funo auxiliar que recebe como entrada uma string com um caminho na rvore de diretrios e apaga todos os ltimos

caracteres at apagar dois “/”. Assim em “/home/alice/projeto/diretorio/” ele retornaria “/home/alice/projeto” efetivamente subindo um nvel na rvore de diretrios.

importante lembrar que no Windows o separador no o “/”, mas o ““”. Ento vamos tratar o separador de forma diferente de acordo com o sistema:

Seo: Funes auxiliares Weaver:

```
void path_up(char *path){
#if !defined(_WIN32)
  char separator = '/';
#else
  char separator = '\\';
#endif
  int erased = 0;
  char *p = path;
  while(*p != '\0') p ++; // Vai at o fim
  while(erased < 2 && p != path){
    p --;
    if(*p == separator) erased ++;
    *p = '\0'; // Apaga
  }
}
```

Note que caso a funo receba uma string que no possua dois “/” em seu nome, acabamos apagando toda a string. Neste programa limitaremos o uso desta funo a strings com caminhos de arquivos que no esto na raz e diretrios diferentes da prpria raz que terminam sempre com “/”, ento no teremos problemas pois a restrio do nmero de barras ser cumprida. Ex: “/etc/” e “/tmp/file.txt”.

5.2. directory_exists: Arquivo existe e diretrio

Para checar se o diretrio `.weaver` existe, definimos `directory_exist(x)` como uma funo que recebe uma string correspondente localizao de um arquivo e que deve retornar 1 se `x` for um diretrio existente, -1 se `x` for um arquivo existente e 0 caso contrrio. Primeiro criamos as macros para no nos esquecermos do que significa cada nmero de retorno:

Seo: Macros do Programa Weaver (continuao):

```
#define NAO_EXISTE             0
#define EXISTE_E_EH_DIRETORIO  1
#define EXISTE_E_EH_ARQUIVO   -1
```

Seo: Funes auxiliares Weaver (continuao):

```
int directory_exist(char *dir){
#if !defined(_WIN32)
  // Unix:
  struct stat s; // Armazena status se um diretrio existe ou no.
  int err; // Checagem de erros
  err = stat(dir, &s); // .weaver existe?
  if(err == -1) return NAO_EXISTE;
  if(S_ISDIR(s.st_mode)) return EXISTE_E_EH_DIRETORIO;
  return EXISTE_E_EH_ARQUIVO;
#else
  // Windows:
  DWORD dwAttrib = GetFileAttributes(dir);
  if(dwAttrib == INVALID_FILE_ATTRIBUTES) return NAO_EXISTE;
```

```
  if(!(dwAttrib & FILE_ATTRIBUTE_DIRECTORY)) return EXISTE_E_EH_ARQUIVO;
  else return EXISTE_E_EH_DIRETORIO;
#endif
}
```

Dependendo de estarmos no Windows ou em sistemas Unix, usamos funes diferentes e vamos precisar de cabealhos diferentes:

Seo: Cabealhos Includos no Programa Weaver:

```
#if !defined(_WIN32)
#include <sys/types.h> // stat, getuid, getpwuid, mkdir
#include <sys/stat.h> // stat, mkdir
#else
#include <windows.h> // GetFileAttributes, ...
#endif
```

5.3. concatenate: Concatena strings

A ltima funo auxiliar da qual precisaremos uma funo para concatenar strings. Ela deve receber um nmero arbitrrio de srings como argumento, mas a ltima string deve ser uma string vazia. E ir retornar a concatenao de todas as strings passadas como argumento.

A funo ir alocar sempre uma nova string, a qual dever ser desalocada antes do programa terminar. Como exemplo, `concatenate("tes", " ", "te", "")` retorna `"tes te"`.

Seo: Funes auxiliares Weaver (continuao):

```
char *concatenate(char *string, ...){
  va_list arguments;
  char *new_string, *current_string = string;
  size_t current_size = strlen(string) + 1;
  char *realloc_return;
  va_start(arguments, string);
  new_string = (char *) malloc(current_size);
  if(new_string == NULL) return NULL;
   // Copia primeira string de acordo com o indicado pelo sistema operacional
#ifdef __OpenBSD__
  strlcpy(new_string, string, current_size);
#else
  strcpy(new_string, string);
#endif
  while(current_string[0] != '\0'){ // Pra quando copiamos o ""
    current_string = va_arg(arguments, char *);
    current_size += strlen(current_string);
    realloc_return = (char *) realloc(new_string, current_size);
    if(realloc_return == NULL){
      free(new_string);
      return NULL;
    }
    new_string = realloc_return;
     // Copia prxima string de acordo com o recomendado pelo sistema
#ifdef __OpenBSD__
    strlcat(new_string, current_string, current_size);
#else
    strcat(new_string, current_string);
```

```
#endif
  }
  return new_string;
}
```

importante lembrarmos que a funo `concatenate` sempre deve receber como ltimo argumento uma string vazia ou teremos um *buffer overflow*. Esta funo perigosa e deve ser usada sempre tomando-se este cuidado.

O uso desta funo requer que usemos o seguinte cabealho:

Seo: Cabealhos Includos no Programa Weaver:

```
#include <string.h> // strcmp, strcat, strcpy, strncmp
#include <stdarg.h> // va_start, va_arg
```

5.4. basename: Obtm o caminho do diretrio de arquivo

Esta funo j existe em sistemas Unix. Dado o caminho completo para um arquivo, ela retorna uma string apenas com o nome do arquivo. Ela no precisa alocar uma nova string, ela retorna um ponteiro para o nome do arquivo dentro do prprio caminho recebido como argumento. Vamos defini-la agora para sistemas Windows:

Seo: Funes auxiliares Weaver (continuao):

```
#if defined(_WIN32)
char *basename(char *path){
  char *p = path;
  char *last_delimiter = NULL;
  while(*p != '\0'){
    if(*p == '\\')
      last_delimiter = p;
    p ++;
  }
  if(last_delimiter != NULL)
    return last_delimiter + 1;
  else
    return path;
}
#endif
```

Mesmo que no precisemos definir esta funo em sistemas Unix, ainda precisamos inclu-la com o cabealho:

Seo: Cabealhos Includos no Programa Weaver:

```
#if !defined(_WIN32)
#include <libgen.h>
#endif
```

5.5. copy_single_file: Copia um nico arquivo para diretrio

A funo `copy_single_file` tenta copiar o arquivo cujo caminho o primeiro argumento para o diretrio cujo caminho o segundo argumento. Se ela conseguir, retorna 1 e retorna 0 caso contrrio.

Seo: Funes auxiliares Weaver (continuao):

```
int copy_single_file(char *file, char *directory){
  int block_size, bytes_read;
  char *buffer, *file_dst;
  FILE *orig, *dst;
```

```
  // Inicializa 'block_size':
        <Seo a ser Inserida: Descobre tamanho do bloco do sistema de arquivos>
  buffer = (char *) malloc(block_size); // Aloca buffer de cpia
  if(buffer == NULL) return 0;
  file_dst = concatenate(directory, "/", basename(file), "");
  if(file_dst == NULL) return 0;
  orig = fopen(file, "r"); // Abre arquivo de origem
  if(orig == NULL){
    free(buffer);
    free(file_dst);
    return 0;
  }
  dst = fopen(file_dst, "w"); // Abre arquivo de destino
  if(dst == NULL){
    fclose(orig);
    free(buffer);
    free(file_dst);
    return 0;
  }
  while((bytes_read = fread(buffer, 1, block_size, orig)) > 0){
    fwrite(buffer, 1, bytes_read, dst); // Copia origem -> buffer -> destino
  }
  fclose(orig);
  fclose(dst);
  free(file_dst);
  free(buffer);
  return 1;
}
```

O mais eficiente que o buffer usado para copiar arquivos tenha o mesmo tamanho do bloco do sistema de arquivos. Para obter o valor correto deste tamanho, usamos o seguinte trecho de cdigo em sistemas Unix:

Seo: Descobre tamanho do bloco do sistema de arquivos:

```
#if !defined(_WIN32)
{
  struct stat s;
  stat(directory, &s);
  block_size = s.st_blksize;
  if(block_size <= 0){
    block_size = 4096;
  }
}
#endif
```

No Windows ns apenas iremos assumir que o tamanho 4 KB:

Seo: Descobre tamanho do bloco do sistema de arquivos (continuao):

```
#if defined(_WIN32)
  block_size = 4096;
#endif
```

5.6. copy_files: Copia todos os arquivos de origem para destino

De posse da funo que copia um s arquivo, precisamos definir uma funo para copiar todos os arquivos de dentro d eum diretrio para outro recursivamente. Isso algo trabalhoso, pois precisamos listar todo o contedo de um diretrio para obter seus arquivos. Como fazer isso varia dependendo do sistema operacional.

Em sistemas Unix a funo usar `readdir` para ler o contedo de arquivos:

Seo: Funes auxiliares Weaver (continuao):

```
#if !defined(_WIN32)
int copy_files(char *orig, char *dst){
  DIR *d = NULL;
  struct dirent *dir;
  d = opendir(orig);
  if(d){
    while((dir = readdir(d)) != NULL){ // Loop para ler cada arquivo
      char *file;
      file = concatenate(orig, "/", dir -> d_name, "");
      if(file == NULL){
        return 0;
      }
#if (defined(__linux__) || defined(_BSD_SOURCE)) && defined(DT_DIR)
      // Se suportamos DT_DIR, no precisamos chamar a funo 'stat':
      if(dir -> d_type == DT_DIR){
#else
      struct stat s;
      int err;
      err = stat(file, &s);
      if(err == -1) return 0;
      if(S_ISDIR(s.st_mode)){
#endif
      // Se concluirmos estar lidando com subdiretrio via 'stat' ou 'DT_DIR':
        char *new_dst;
        new_dst = concatenate(dst, "/", dir -> d_name, "");
        if(new_dst == NULL){
          return 0;
        }
        if(strcmp(dir -> d_name, ".") && strcmp(dir -> d_name, "..")){
          if(directory_exist(new_dst) == NAO_EXISTE) mkdir(new_dst, 0755);
          if(copy_files(file, new_dst) == 0){
            free(new_dst);
            free(file);
            closedir(d);
            return 0;
          }
        }
        free(new_dst);
      }
      else{
        // Se concluimos estar diante de um arquivo usual:
        if(copy_single_file(file, dst) == 0){
          free(file);
          closedir(d);
```

```
          return 0;
        }
      }
    free(file);
    } // Fim do loop para ler cada arquivo
    closedir(d);
  }
  return 1;
}
#endif
```

E isso requer inserir o cabealho:

Seo: Cabealhos Includos no Programa Weaver:

```
#if !defined(_WIN32)
#include <dirent.h> // readdir, opendir, closedir
#endif
```

No Windows, no necessrio inserir nenhum cabealho novo que j no est inserido. A definio da funo fica assim:

Seo: Funes auxiliares Weaver (continuao):

```
#if defined(_WIN32)
int copy_files(char *orig, char *dst){
  char *path, *search_path;
  WIN32_FIND_DATA file;
  HANDLE dir = NULL;
  search_path = concatenate(orig, "\\*", "");
  if(search_path == NULL)
    return 0;
  dir = FindFirstFile(search_path, &file);
  if(dir != INVALID_HANDLE_VALUE){
    // The first file shall be '.' and should be safely ignored
    do{
      if(strcmp(file.cFileName, ".") && strcmp(file.cFileName, "..")){
        path = concatenate(orig, "\\", file.cFileName, "");
        if(path == NULL){
Ψ  free(search_path);
          return 0;
        }
        if(file.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY){
          char *dst_path;
          dst_path = concatenate(dst, "\\", file.cFileName, "");
          if(directory_exist(dst_path) == NAO_EXISTE)
            CreateDirectoryA(dst_path, NULL);
          if(copy_files(path, dst_path) == 0){
            free(dst_path);
            free(path);
            FindClose(dir);
Ψ    free(search_path);
            return 0;
          }
          free(dst_path);
```

```
        }
        else{ // file
          if(copy_single_file(path, dst) == 0){
            free(path);
            FindClose(dir);
Ψ    free(search_path);
            return 0;
          }
        }
        free(path);
      }
    }while(FindNextFile(dir, &file));
  }
  free(search_path);
  FindClose(dir);
  return 1;
}
#endif
```

5.7. write_copyright: Escreve mensagem de copyright em arquivo

Por padro, projetos Weaver utilizam a licena GNU GPL3. Como cdigos sob esta licena so copiados e usados estaticamente em novos projetos, eles precisam necessariamente ter uma licena igual ou compatvel.

O cdigo bastante simples e requer apenas alguns parmetros com o nome do autor e ano atual:

Seo: Funes auxiliares Weaver (continuao):

```
void write_copyright(FILE *fp, char *author_name, char *project_name, int year){
  char license[] = "/*\nCopyright (c) %s, %d\n\nThis file is part of %s.\n\n%s\
 is free software: you can redistribute it and/or modify\nit under the terms of\
 the GNU Affero General Public License as published by\nthe Free Software\
 Foundation, either version 3 of the License, or\n(at your option) any later\
 version.\n\n\
%s is distributed in the hope that it will be useful,\nbut WITHOUT ANY\
  WARRANTY; without even the implied warranty of\nMERCHANTABILITY or FITNESS\
  FOR A PARTICULAR PURPOSE.  See the\nGNU Affero General Public License for more\
  details.\n\nYou should have received a copy of the GNU Affero General Public License\
\nalong with %s. If not, see <http://www.gnu.org/licenses/>.\n*/\n\n";
  fprintf(fp, license, author_name, year, project_name, project_name,
          project_name, project_name);
}
```

5.8. create_dir: Cria novos diretrios

Esta a funo responsvel por criar uma lista de diretrios. Isso algo bastante simples, mas deve ficar encapsulado em sua prpria funo por causa das diferenas entre Sistemas Operacionais sobre como realizar a tarefa.

Esta funo deve receber uma lista varivel de strings como argumento, sendo que o ltimo argumento precisa ser uma string vazia ou NULL. Para cada argumento representando um caminho, a funo criar o diretrio no caminho especificado. Por padro, usaremos como separador em caminhos ser o "/", de modo que isso far a funo funcionar tanto em sistemas Unix como no Windows, onde a sua funo `CreateDirectoryA` entende caminhos passados com o

separador Unix.

Em sistemas Unix ns precisamos tambm especificar as permisses mximas que o diretrio ter em termos de leitura, escrita e execuo. Tais permisses podem ser mais restritas de acordo com a configurao do sistema. No Windows, que tem um sistema de permisses mais hierrquica, ns apenas herdamos as permisses do diretrio pai.

Em caso de erro, ns retornaremos -1. Se tudo deu certo, retornamos 1.

A definio de nossa funo ser ento:

Seo: Funes auxiliares Weaver (continuao):

```
int create_dir(char *string, ...){
  char *current_string;
  va_list arguments;
  va_start(arguments, string);
  int err = 1;
  current_string = string;
  while(current_string != NULL && current_string[0] != '\0' && err != -1){
#if !defined(_WIN32)
    err = mkdir(current_string, S_IRWXU | S_IRWXG | S_IROTH);
#else
    if(!CreateDirectoryA(current_string, NULL))
      err = -1;
#endif
    current_string = va_arg(arguments, char *);
  }
  return err;
}
```

5.9. append_file: Concatena um arquivo no outro

Esta ser uma funo diferente, pois ela ser mais focada em resolver de maneira eficiente um nico caso de uso do que apresentar uma interface intuitiva e consistente. Ela ir receber como entrada um ponteiro para um arquivo j aberto (iremos chamar esta funo depois de j termos aberto o arquivo para escrever o copyright) e receber dois outros argumentos: o diretrio em que est o arquivo de origem e o nome do arquivo de origem, separados em duas strings. Dessa forma, o trabalho de concatenar as duas coisas fica com esta funo, no com quem a est invocando.

Sua definio ser:

Seo: Funes auxiliares Weaver (continuao):

```
int append_file(FILE *fp, char *dir, char *file){
  int block_size, bytes_read;
  char *buffer, *directory = ".";
  char *path = concatenate(dir, file, "");
  if(path == NULL) return 0;
  FILE *origin;
        <Seo a ser Inserida: Descobre tamanho do bloco do sistema de arquivos>
  buffer = (char *) malloc(block_size);
  if(buffer == NULL){
    free(path);
    return 0;
  }
  origin = fopen(path, "r");
  if(origin == NULL){
```

```
    free(buffer);
    free(path);
    return 0;
  }
  while((bytes_read = fread(buffer, 1, block_size, origin)) > 0){
    fwrite(buffer, 1, bytes_read, fp);
  }
  fclose(origin);
  free(buffer);
  free(path);
  return 1;
}
```

6. Inicializao das Variveis

6.1. inside_weaver_directory e project_path: Onde estamos

A primeira das variveis `inside_weaver_directory` , que deve valer `false` se o programa foi invocado de fora de um diretrio de projeto Weaver e `true` caso contrrio.

Como definir se estamos em um diretrio que pertence um projeto Weaver? Simples. So diretrios que contm dentro de si ou em um diretrio ancestral um diretrio oculto chamado `.weaver`. Caso encontremos este diretrio oculto, tambm podemos aproveitar e ajustar a varivel `project_path` para apontar para o local onde ele est. Se no o encontrarmos, estaremos fora de um diretrio Weaver e no precisamos mudar nenhum valor das duas variveis, pois elas devero permanecer com o valor padro `NULL` .

Em suma, o que precisamos de um loop com as seguintes caractersticas:

Invariantes: A varivel `complete_path` deve sempre possuir o caminho completo do diretrio `.weaver` se ele existisse no diretrio atual.

Inicializao: Inicializamos tanto o `complete_path` para serem vlidos de acordo com o diretrio em que o programa invocado.

Manuteno: Em cada iterao do loop ns verificamos se encontramos uma condio de finalizao. Caso contrrio, subimos para o diretrio pai do qual estamos, sempre atualizando as variveis para que o invariante continue vlido.

Finalizao: Interrompemos a execuo do loop se uma das trs condies ocorrerem:

a) `complete_path == "/.weaver"` : Neste caso no podemos subir mais na rvore de diretrios, pois estamos na raiz do sistema de arquivos. No encontramos um diretrio `.weaver`. Isso significa que no estamos dentro de um projeto Weaver.

b) `complete_path == "C:\\.weaver"` : A letra inicial pode no ser um “C”. De qualquer forma, estamos na raz do sistema dos arquivos e no podemos subir mais como no caso acima. Com a diferena de estarmos no Windows.

c) `complete_path == "./.weaver"` e tal arquivo existe e diretrio: Neste caso encontramos um diretrio `.weaver` e descobrimos que estamos dentro de um projeto Weaver. Podemos ento atualizar a varivel `project_path` para o diretrio em que paramos.

O cdigo de inicializao destas variveis ser ento:

Seo: Inicializao:

```
char *path = NULL, *complete_path = NULL;
#if !defined(_WIN32)
path = getcwd(NULL, 0); // Unix
#else
{ // Windows
  DWORD bsize;
  bsize = GetCurrentDirectory(0, NULL);
  path = (char *) malloc(bsize);
  GetCurrentDirectory(bsize, path);
```

```
}
#endif
if(path == NULL) W_ERROR();
complete_path = concatenate(path, "/.weaver", "");
free(path);
if(complete_path == NULL) W_ERROR();
```

Para obtermos o diretrio atual, vamos precisar do cabealho:

Seo: Cabealhos Includos no Programa Weaver:

```
#if !defined(_WIN32)
#include <unistd.h> // get_current_dir_name, getcwd, stat, chdir, getuid
#endif
```

Agora iniciamos um loop que terminar quando `complete_path` for igual `/.weaver` (chegamos no fim da rvore de diretrios e no encontramos nada) ou quando realmente existir o diretrio `.weaver/` no diretrio examinado. E no fim do loop, sempre vamos para o diretrio-pai do qual estamos:

Seo: Inicializao (continuao):

```
{
  size_t tmp_size = strlen(complete_path);
  // Testa se chegamos ao fim:
  while(strcmp(complete_path, "/.weaver") &&
Ψstrcmp(complete_path, "\\.weaver") &&
Ψstrcmp(complete_path + 1, ":\\.weaver")){
    if(directory_exist(complete_path) == EXISTE_E_EH_DIRETORIO){
      inside_weaver_directory = true;
      complete_path[strlen(complete_path) - 7] = '\0'; // Apaga o '.weaver'
      project_path = concatenate(complete_path, "");
      if(project_path == NULL){ free(complete_path); W_ERROR(); }
      break;
    }
    else{
      path_up(complete_path);
#ifdef __OpenBSD__
      strlcat(complete_path, "/.weaver", tmp_size);
#else
      strcat(complete_path, "/.weaver");
#endif
    }
  }
  free(complete_path);
}
```

Como alocamos memria para `project_path` armazenar o endereo do projeto atual se estamos em um projeto Weaver, no final do programa teremos que desalocar a memria:

Seo: Finalizao:

```
if(project_path != NULL) free(project_path);
```

6.2. weaver_version_major e weaver_version_minor: Verso do Programa

Para descobrirmos a verso atual do Weaver que temos, basta consultar o valor presente na macro `VERSION` . Ento, obtemos o nmero de verso maior e menor que esto separados por um ponto (se existirem). Note que se no houver um ponto no nome da verso, ento ela uma verso de testes. Mesmo neste caso o cdigo abaixo vai funcionar, pois a funo `atoi` iria retornar 0 nas duas invocaes por encontrar

respectivamente uma string sem dgito algum e um fim de string sem contedo:

Seo: Inicializao (continuao):

```
{
  char *p = VERSION;
  while(*p != '.' && *p != '\0') p ++;
  if(*p == '.') p ++;
  weaver_version_major = atoi(VERSION);
  weaver_version_minor = atoi(p);
}
```

6.3. project_version_major e project_version_minor: Verso do Projeto

Se estamos dentro de um projeto Weaver, temos que inicializar informao sobre qual verso do Weaver foi usada para atualiz-lo pela ltima vez. Isso pode ser obtido lendo o arquivo *.weaver/version* localizado dentro do diretrio Weaver. Se no estamos em um diretrio Weaver, no precisamos inicializar tais valores. O nmero de verso maior e menor separado por um ponto.

Seo: Inicializao (continuao):

```
if(inside_weaver_directory){
  FILE *fp;
  char *p, version[10];
  char *file_path = concatenate(project_path, ".weaver/version", "");
  if(file_path == NULL) W_ERROR();
  fp = fopen(file_path, "r");
  free(file_path);
  if(fp == NULL) W_ERROR();
  p = fgets(version, 10, fp);
  if(p == NULL){ fclose(fp); W_ERROR(); }
  while(*p != '.' && *p != '\0') p ++;
  if(*p == '.') p ++;
  project_version_major = atoi(version);
  project_version_minor = atoi(p);
  fclose(fp);
}
```

6.4. have_arg, argument e argument2: Argumentos de Invocao

Uma das variveis mais fceis e triviais de se inicializar. Basta consultar `argc` e `argv`.

Seo: Inicializao (continuao):

```
have_arg = (argc > 1);
if(have_arg) argument = argv[1];
if(argc > 2) argument2 = argv[2];
```

6.5. arg_is_path: Se argumento diretrio

Agora temos que verificar se no caso de termos um argumento, se ele um caminho para um projeto Weaver existente ou no. Para isso, checamos se ao concatenarmos `/.weaver` no argumento encontramos o caminho de um diretrio existente ou no.

Seo: Inicializao (continuao):

```
if(have_arg){
  char *buffer = concatenate(argument, "/.weaver", "");
```

```
  if(buffer == NULL) W_ERROR();
  if(directory_exist(buffer) == EXISTE_E_EH_DIRETORIO){
    arg_is_path = 1;
  }
  free(buffer);
}
```

6.6. shared_dir: Onde arquivos esto instalados

A varivel `shared_dir` dever conter onde esto os arquivos compartilhados da instalao de Weaver. Tais arquivos so as prprias bibliotecas a serem inseridas estaticamente e modelos de cdigo fonte. Se existir a macro passada durante a compilao `WEAVER_DIR`, este ser o caminho em que esto os arquivos. Caso contrrio, assumiremos o valor padro de `/usr/local/share/weaver` em sistemas baseados em Unix e o local apontado pela varivel de ambiente ProgramFiles em ambientes Windows.

Seo: Inicializao (continuao):

```
{
#ifdef WEAVER_DIR
  shared_dir = concatenate(WEAVER_DIR, "");
#else
#if !defined(_WIN32)
  shared_dir = concatenate("/usr/local/share/weaver/", ""); // Unix
#else
  { // Windows
    char *temp_buf = NULL;
    DWORD bsize = GetEnvironmentVariable("ProgramFiles", temp_buf, 0);
    temp_buf = (char *) malloc(bsize);
    GetEnvironmentVariable("ProgramFiles", temp_buf, bsize);
    shared_dir = concatenate(temp_buf, "\\weaver\\", "");
    free(temp_buf);
  }
#endif
#endif
  if(shared_dir == NULL) W_ERROR();
}
```

Com isso damos poder durante a compilao para determinar onde os dados do motor Weaver sero armazenados no sistema. Algo mais comum de ser alterado em sistemas Unix que no Windows, onde espera-se que os programas sejam armazenados no mesmo lugar.

No Windows o cdigo mais longo principalmente por termos que determinar manualmente o nome do local padro de se armazenar os programas. O endereo pode variar de acordo com o idioma do sistema, com a unidade de volume em que ele est ou com o fato do programa ter sido compilado em mquina com 32 ou 64 bits.

No fim do programa devemos desalocar a memria alocada para `shared_dir` :

Seo: Finalizao (continuao):

```
if(shared_dir != NULL) free(shared_dir);
```

6.7. arg_is_valid_project: Se o argumento um nome de projeto

A prxima questo que deve ser averiguada se o que recebemos como argumento, caso haja argumento, pode ser o nome de um projeto Weaver vlido ou no. Para isso, trs condies precisam ser satisfeitas:

1) O nome base do projeto deve ser formado somente por caracteres alfanumricos e underline (embora uma barra possa aparecer para passar o caminho completo de um projeto).

2) No pode existir um arquivo com o mesmo nome do projeto no local indicado para a criao.

3) O projeto no pode ter o nome de nenhum arquivo que costuma ficar no diretrio base de um projeto Weaver (como "Makefile"). Do contrrio, na hora da compilao comandos como "`gcc game.c -o Makefile`" poderiam ser executados e sobrescreveriam arquivos importantes.

Para isso, usamos o seguinte cdigo:

Seo: Inicializao (continuao):

```
if(have_arg && !arg_is_path){
  char *buffer;
  char *base = basename(argument);
  int size = strlen(base);
  int i;
  // Checando caracteres invlidos no nome:
  for(i = 0; i < size; i ++){
    if(!isalnum(base[i]) && base[i] != '_'){
      goto NOT_VALID;
    }
  }
  // Checando se arquivo existe:
  if(directory_exist(argument) != NAO_EXISTE){
    goto NOT_VALID;
  }
  // Checando se conflita com arquivos de compilao:
  buffer = concatenate(shared_dir, "project/", base, "");
  if(buffer == NULL) W_ERROR();
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID;
  }
  free(buffer);
  arg_is_valid_project = true;
}
NOT_VALID:
```

Para podermos checar se um caractere alfanumrico, incluimos a seguinte biblioteca:

Seo: Cabealhos Includos no Programa Weaver:

```
#include <ctype.h> // isalnum
```

6.8. arg_is_valid_module: Se o argumento pode ser um nome de mdulo

Checar se o argumento que recebemos pode ser um nome vlido para um mdulo s faz sentido se estivermos dentro de um diretrio Weaver e se um argumento estiver sendo passado. Neste caso, o argumento um nome vlido se ele contiver apenas caracteres alfanumricos, underline e se no existir no projeto um arquivo `.c` ou `.h` em `src/` que tenha o mesmo nome do argumento passado:

Seo: Inicializao (continuao):

```
if(have_arg && inside_weaver_directory){
  char *buffer;
```

```
  int i, size;
  size = strlen(argument);
  // Checando caracteres invlidos no nome:
  for(i = 0; i < size; i ++){
    if(!isalnum(argument[i]) && argument[i] != '_'){
      goto NOT_VALID_MODULE;
    }
  }
  // Checando por conflito de nomes:
  buffer = concatenate(project_path, "src/", argument, ".c", "");
  if(buffer == NULL) W_ERROR();
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID_MODULE;
  }
  buffer[strlen(buffer) - 1] = 'h';
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID_MODULE;
  }
  free(buffer);
  arg_is_valid_module = true;
}
NOT_VALID_MODULE:
```

6.9. arg_is_valid_plugin: Se o argumento pode ser um nome de plugin

Para que um argumento seja um nome vlido para plugin, ele deve ser composto s por caracteres alfanumricos ou underline e no existir no diretrio `plugin` um arquivo com a extenso `.c` de mesmo nome. Tambm precisamos estar naturalmente, em um diretrio Weaver.

Seo: Inicializao (continuao):

```
if(argument2 != NULL && inside_weaver_directory){
  int i, size;
  char *buffer;
  size = strlen(argument2);
  // Checando caracteres invlidos no nome:
  for(i = 0; i < size; i ++){
    if(!isalnum(argument2[i]) && argument2[i] != '_'){
      goto NOT_VALID_PLUGIN;
    }
  }
  // Checando se j existe plugin com mesmo nome:
  buffer = concatenate(project_path, "plugins/", argument2, ".c", "");
  if(buffer == NULL) W_ERROR();
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID_PLUGIN;
  }
  free(buffer);
  arg_is_valid_plugin = true;
}
```

```
NOT_VALID_PLUGIN:
```

6.10. arg_is_valid_function: Se o argumento pode ser um nome de funo de loop principal

Para que essa varivel seja verdadeira, preciso existir um segundo argumento e ele deve ser formado somente por caracteres alfanumricos ou underline. Alm disso, o primeiro caractere precisa ser uma letra e ele no pode ter o mesmo nome de alguma palavra reservada em C.

Seo: Inicializao (continuao):

```
if(argument2 != NULL && inside_weaver_directory &&
   !strcmp(argument, "--loop")){
  int i, size;
  char *buffer;
  // Primeiro caractere no pode ser dgito
  if(isdigit(argument2[0]))
    goto NOT_VALID_FUNCTION;
  size = strlen(argument2);
  // Checando caracteres invlidos no nome:
  for(i = 0; i < size; i ++){
    if(!isalnum(argument2[i]) && argument2[i] != '_'){
      goto NOT_VALID_PLUGIN;
    }
  }
  // Checando se existem arquivos com o nome indicado:
  buffer = concatenate(project_path, "src/", argument2, ".c", "");
  if(buffer == NULL) W_ERROR();
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID_FUNCTION;
  }
  buffer[strlen(buffer)-1] = 'h';
  if(directory_exist(buffer) != NAO_EXISTE){
    free(buffer);
    goto NOT_VALID_FUNCTION;
  }
  free(buffer);
  // Checando se recebemos como argumento uma palavra reservada em C:
  if(!strcmp(argument2, "auto") || !strcmp(argument2, "break") ||
     !strcmp(argument2, "case") || !strcmp(argument2, "char") ||
     !strcmp(argument2, "const") || !strcmp(argument2, "continue") ||
     !strcmp(argument2, "default") || !strcmp(argument2, "do") ||
     !strcmp(argument2, "int") || !strcmp(argument2, "long") ||
     !strcmp(argument2, "register") || !strcmp(argument2, "return") ||
     !strcmp(argument2, "short") || !strcmp(argument2, "signed") ||
     !strcmp(argument2, "sizeof") || !strcmp(argument2, "static") ||
     !strcmp(argument2, "struct") || !strcmp(argument2, "switch") ||
     !strcmp(argument2, "typedef") || !strcmp(argument2, "union") ||
     !strcmp(argument2, "unsigned") || !strcmp(argument2, "void") ||
     !strcmp(argument2, "volatile") || !strcmp(argument2, "while") ||
     !strcmp(argument2, "double") || !strcmp(argument2, "else") ||
     !strcmp(argument2, "enum") || !strcmp(argument2, "extern") ||
```

```
     !strcmp(argument2, "float") || !strcmp(argument2, "for") ||
     !strcmp(argument2, "goto") || !strcmp(argument2, "if"))
    goto NOT_VALID_FUNCTION;
  arg_is_valid_function = true;
}
NOT_VALID_FUNCTION:
```

6.11. author_name: Nome do criador do cdigo

A varivel `author_name` deve conter o nome do usurio que est invocando o programa. Esta informao til para gerar uma mensagem de Copyright nos arquivos de cdigo fonte de novos mdulos.

Isso ser feito de maneira diferente em sistemas Unix e Windows. Em sistemas Unix, comeamos obtendo o seu UID. De posse dele, obtemos todas as informaes de login com um `getpwuid`. Se o usurio tiver registrado um nome em `/etc/passwd`, obtemos tal nome na estrutura retornada pela funo. Caso contrrio, assumiremos o login como sendo o nome:

Seo: Inicializao (continuao):

```
#if !defined(_WIN32)
{
  struct passwd *login;
  int size;
  char *string_to_copy;
  login = getpwuid(getuid()); // Obtm dados de usurio
  if(login == NULL) W_ERROR();
  size = strlen(login -> pw_gecos);
  if(size > 0)
    string_to_copy = login -> pw_gecos;
  else
    string_to_copy = login -> pw_name;
  size = strlen(string_to_copy);
  author_name = (char *) malloc(size + 1);
  if(author_name == NULL) W_ERROR();
#ifdef __OpenBSD__
  strlcpy(author_name, string_to_copy, size + 1);
#else
  strcpy(author_name, string_to_copy);
#endif
}
#endif
```

No Windows, o nome pode ser obtido com a funo `GetUserNameExA`. Na primeira invocao tentamos obter o tamanho do buffer necessrio para armazenarmos o nome e na segunda obtemos o nome em si. Em caso de erro, usamos a funo mais antiga `GetUserNameA` que vai retornar um nome de usurio simples ao invs de tentar obter o nome completo, e para isso alocamos um espao para o maior nome de usurio vlido no sistema.

Seo: Inicializao (continuao):

```
#if defined(_WIN32)
{
  int size = 0;
  GetUserNameExA(NameDisplay, author_name, &size);
  if(GetLastError() == ERROR_MORE_DATA){
    if(size == 0)
```

```
      size = 64;
    author_name = (char *) malloc(size);
    if(GetUserNameExA(NameDisplay, author_name, &size) == 0){
      size = UNLEN + 1;
      author_name = (char *) malloc(size);
      GetUserNameA(author_name, &size);
    }
  }
  else{
    size = UNLEN + 1;
    author_name = (char *) malloc(size);
    GetUserNameA(author_name, &size);
  }
}
#endif
```

Depois, precisaremos desalocar a memria ocupada por `author_name` :

Seo: Finalizao (continuao):

```
if(author_name != NULL) free(author_name);
```

Para que o cdigo funcione, devemos inserir uma biblioteca diferente dependendo de estarmos em sistemas Unix (para ter `getpwuid`) ou em sistemas Windows (para obtermos uma enumerao com diferentes formatos de nomes):

Seo: Cabealhos Includos no Programa Weaver:

```
#if !defined(_WIN32)
#include <pwd.h> // getpwuid
#else
#define SECURITY_WIN32
#include <Security.h>
#include <Lmcons.h>
#endif
```

6.12. project_name: Nome do projeto

S faz sendido falarmos no nome do projeto se estivermos dentro de um projeto Weaver. Neste caso, o nome do projeto pode ser encontrado em um dos arquivos do diretrio base de tal projeto em `.weaver/name`:

Seo: Inicializao (continuao):

```
if(inside_weaver_directory){
  FILE *fp;
  char *c;
#if !defined(_WIN32)
  char *filename = concatenate(project_path, ".weaver/name", "");
#else
  char *filename = concatenate(project_path, ".weaver\name", "");
#endif
  if(filename == NULL) W_ERROR();
  project_name = (char *) malloc(256);
  if(project_name == NULL){
    free(filename);
    W_ERROR();
  }
```

```
  fp = fopen(filename, "r");
  if(fp == NULL){
    free(filename);
    W_ERROR();
  }
  c = fgets(project_name, 256, fp);
  fclose(fp);
  free(filename);
  if(c == NULL) W_ERROR();
  project_name[strlen(project_name)-1] = '\0';
  project_name = realloc(project_name, strlen(project_name) + 1);
  if(project_name == NULL) W_ERROR();
}
```

Depois, precisaremos desalocar a memria ocupada por `project_name` :

Seo: Finalizao (continuao):

```
if(project_name != NULL) free(project_name);
```

6.13. year: Ano atual

O ano atual trivial de descobrir usando a funo `localtime` , independente do sistema operacional:

Seo: Inicializao (continuao):

```
{
  time_t current_time;
  struct tm *date;
  time(&current_time);
  date = localtime(&current_time);
  year = date -> tm_year + 1900;
}
```

O nico pr-requisito incluirmos antes a biblioteca com funes de tempo:

Seo: Cabealhos Includos no Programa Weaver:

```
#include <time.h> // localtime, time
```

7. Casos de Uso

7.1. Imprimir ajuda de criao de projeto

O primeiro caso de uso sempre ocorre quando Weaver invocado fora de um diretrio de projeto e a invocao sem argumentos ou com argumento `--help`. Nesse caso assumimos que o usurio no sabe bem como usar o programa e imprimimos uma mensagem de ajuda. A mensagem de ajuda ter uma forma semelhante a esta:

```
    .  .   You are outside a Weaver Directory.
   ./  \.  The following command uses are available:
   \\  //
   \\()//  weaver
   .={}=.      Print this message and exits.
  / /‘’\ \
  ‘ \  / ’  weaver PROJECT_NAME
     ‘’         Creates a new Weaver Directory with a new
                project.
```

O que feito com o cdigo abaixo:

Seo: Caso de uso 1: Imprimir ajuda (criar projeto):

```
if(!inside_weaver_directory && (!have_arg || !strcmp(argument, "--help"))){
  printf("    .  .     You are outside a Weaver Directory.\n"
  "   .|  |.    The following command uses are available:\n"
  "   ||  ||\n"
  "   \\\\()//  weaver\n"
  "   .={}=.      Print this message and exits.\n"
  "  / /‘’\\ \\\n"
  "  ‘ \\  / ’  weaver PROJECT_NAME\n"
  "     ‘’        Creates a new Weaver Directory with a new\n"
  "               project.\n");
  END();
}
```

7.2. Imprimir ajuda de gerenciamento

O segundo caso de uso tambm bastante simples. Ele invocado quando j estamos dentro de um projeto Weaver e invocamos Weaver sem argumentos ou com um `--help`. Assumimos neste caso que o usurio quer instrues sobre a criao de um novo mdulo. A mensagem que imprimiremos semelhante esta:

```
       \              You are inside a Weaver Directory.
        \______/      The following command uses are available:
        /\____/\
       / /\__/\ \       weaver
    __/_/_/\/\_\_\___     Prints this message and exits.
      \ \ \/\/ / /
       \ \/__\/ /       weaver NAME
        \/____\/          Creates NAME.c and NAME.h, updating
        /      \          the Makefile and headers
       /

                         weaver --loop NAME
                          Creates a new main loop in a new file src/NAME.c

                         weaver --plugin NAME
                          Creates new plugin in plugin/NAME.c

                         weaver --shader NAME
                          Creates a new shader directory in shaders/
```

O que obtido com o cdigo:

Seo: Caso de uso 2: Imprimir ajuda de gerenciamento:

```
if(inside_weaver_directory && (!have_arg || !strcmp(argument, "--help"))){
  printf("       \\               You are inside a Weaver Directory.\n"
  "        \\______/        The following command uses are available:\n"
  "        /\\____/\\\n"
  "       / /\\__/\\ \\       weaver\n"
  "    __/_/_/\\/\\_\\_\\___     Prints this message and exits.\n"
  "      \\ \\ \\/\\/ / /\n"
  "       \\ \\/__\\/ /       weaver NAME\n"
  "        \\/____\\/          Creates NAME.c and NAME.h, updating\n"
  "        /      \\          the Makefile and headers\n"
```

```
  "       /\n"
  "                          weaver --loop NAME\n"
  "                           Creates a new main loop in a new file src/NAME.c\n\n"
  "                          weaver --plugin NAME\n"
  "                           Creates a new plugin in plugin/NAME.c\n\n"
  "                          weaver --shader NAME\n"
  "                           Creates a new shader directory in shaders/\n");
  END();
}
```

7.3. Mostrar a verso instalada de Weaver

Um caso de uso ainda mais simples. Ocorrer toda vez que o usurio invocar Weaver com o argumento `--version`:

Seo: Caso de uso 3: Mostrar verso:

```
if(have_arg && !strcmp(argument, "--version")){
  printf("Weaver\t%s\n", VERSION);
  END();
}
```

7.4. Atualizar projetos Weaver j existentes

Este caso de uso ocorre quando o usurio passar como argumento para Weaver um caminho absoluto ou relativo para um diretrio Weaver existente. Assumimos ento que ele deseja atualizar o projeto passado como argumento. Talvez o projeto tenha sido feito com uma verso muito antiga do motor e ele deseja que ele passe a usar uma verso mais nova da API.

Naturalmente, isso s ser feito caso a verso de Weaver instalada seja superior verso do projeto ou se a verso de Weaver instalada for uma verso instvel para testes. Entende-se neste caso que o usurio deseja testar a verso experimental de Weaver no projeto. Fora isso, no possvel fazer *downgrades* de projetos, passando da verso 0.2 para 0.1, por exemplo.

Verses experimentais sempre so identificadas como tendo um nome formado somente por caracteres alfabticos. Verses estveis sero sempre formadas por um ou mais dgitos, um ponto e um ou mais dgitos (o nmero de verso maior e menor). Como o nmero de verso interpretado com um `atoi`, isso significa que se estamos usando uma verso experimental, ento o nmero de verso maior e menor sero sempre identificados como zero.

Projetos em verses experimentais de Weaver sempre sero atualizados, independente da verso ser mais antiga ou mais nova.

Uma atualizao consiste em copiar todos os arquivos que esto no diretrio de arquivos compartilhados Weaver dentro de `project/src/weaver` para o diretrio `src/weaver` do projeto em questo. Para isso podemos contar com as funes de cpia de arquivos definidos na seo de funes auxiliares.

Seo: Caso de uso 4: Atualizar projeto Weaver:

```
if(arg_is_path){
  if((weaver_version_major == 0 && weaver_version_minor == 0) ||
     (weaver_version_major > project_version_major) ||
     (weaver_version_major == project_version_major &&
      weaver_version_minor > project_version_minor)){
    char *buffer, *buffer2;
    // |buffer| <- SHARED_DIR/project/src/weaver
    buffer = concatenate(shared_dir, "project/src/weaver/", "");
    if(buffer == NULL) W_ERROR();
    // |buffer2| <- PROJECT_DIR/src/weaver/
    buffer2 = concatenate(argument, "/src/weaver/", "");
    if(buffer2 == NULL){
```

```
      free(buffer);
      W_ERROR();
    }
    if(copy_files(buffer, buffer2) == 0){
      free(buffer);
      free(buffer2);
      W_ERROR();
    }
    free(buffer);
    free(buffer2);
  }
  END();
}
```

7.5. Adicionando um mdulo ao projeto Weaver

Se estamos dentro de um diretrio de projeto Weaver, e o programa recebeu um argumento, ento estamos inserindo um novo mdulo no nosso jogo. Se o argumento um nome vlido, podemos fazer isso. Caso contrrio,devemos imprimir uma mensagem de erro e sair.

Criar um mdulo basicamente envolve:

a) Criar arquivos `.c` e `.h` base, deixando seus nomes iguais ao nome do mdulo criado.

b) Adicionar em ambos um cdigo com copyright e licenciamento com o nome do autor, do projeto e ano.

c) Adicionar no `.h` cdigo de macro simples para evitar que o cabealho seja inserido mais de uma vez e fazer com que o `.c` inclua o `.h` dentro de si.

d) Fazer com que o `.h` gerado seja inserido em `src/includes.h` e assim suas estruturas sejam acessveis de todos os outros mdulos do jogo.

O cdigo para isso :

Seo: Caso de uso 5: Criar novo mdulo:

```
if(inside_weaver_directory && have_arg &&
   strcmp(argument, "--plugin") && strcmp(argument, "--shader") &&
   strcmp(argument, "--loop")){
  if(arg_is_valid_module){
    char *filename;
    FILE *fp;
    // Criando modulo.c
    filename = concatenate(project_path, "src/", argument, ".c", "");
    if(filename == NULL) W_ERROR();
    fp = fopen(filename, "w");
    if(fp == NULL){
      free(filename);
      W_ERROR();
    }
    write_copyright(fp, author_name, project_name, year);
    fprintf(fp, "#include \"%s.h\"", argument);
    fclose(fp);
    filename[strlen(filename)-1] = 'h'; // Criando modulo.h
    fp = fopen(filename, "w");
    if(fp == NULL){
      free(filename);
      W_ERROR();
    }
```

```
    write_copyright(fp, author_name, project_name, year);
    fprintf(fp, "#ifndef _%s_h_\n", argument);
    fprintf(fp, "#define _%s_h_\n\n#include \"weaver/weaver.h\"\n",
            argument);
    fprintf(fp, "#include \"includes.h\"\n\n#endif");
    fclose(fp);
    free(filename);
```

```
    // Atualizando src/includes.h para inserir modulo.h:
    fp = fopen("src/includes.h", "a");
    fprintf(fp, "#include \"%s.h\"\n", argument);
    fclose(fp);
  }
  else{
    fprintf(stderr, "ERROR: This module name is invalid.\n");
    return_value = 1;
  }
  END();
}
```

7.6. Criar novo projeto

Criar um novo projeto Weaver consiste em criar um novo diretrio com o nome do projeto, copiar para l tudo o que est no diretrio `project` do diretrio de arquivos compartilhados e criar um diretrio `.weaver` com os dados do projeto. Alm disso, criamos um `src/game.c` e `src/game.h` adicionando o comentrio de Copyright neles e copiando a estrutura bsica dos arquivos do diretrio compartilhado `basefile.c` e `basefile.h`. Tambm criamos um `src/includes.h` que por hora estar vazio, mas ser modificado na criao de futuros mdulos.

Seo: Caso de uso 6: Criar novo projeto:

```
if(! inside_weaver_directory && have_arg){
  if(arg_is_valid_project){
    int err;
    char *dir_name;
    FILE *fp;
    err = create_dir(argument, NULL);
    if(err == -1) W_ERROR();
#if !defined(_WIN32) //cd
    err = chdir(argument);
#else
    err = _chdir(argument);
#endif
    if(err == -1) W_ERROR();
    err = create_dir(".weaver", "conf", "tex", "src", "src/weaver",
                     "fonts", "image", "sound", "models", "music",
                     "plugins", "src/misc", "src/misc/sqlite",
                     "compiled_plugins", "shaders", "");
    if(err == -1) W_ERROR();
    dir_name = concatenate(shared_dir, "project", "");
    if(dir_name == NULL) W_ERROR();
    if(copy_files(dir_name, ".") == 0){
      free(dir_name);
      W_ERROR();
```

```
    }
    free(dir_name); //Criando arquivo com nmero de verso:
    fp = fopen(".weaver/version", "w");
    fprintf(fp, "%s\n", VERSION);
    fclose(fp); // Criando arquivo com nome de projeto:
    fp = fopen(".weaver/name", "w");
    fprintf(fp, "%s\n", basename(argv[1]));
    fclose(fp);
    fp = fopen("src/game.c", "w");
    if(fp == NULL) W_ERROR();
    write_copyright(fp, author_name, argument, year);
    if(append_file(fp, shared_dir, "basefile.c") == 0) W_ERROR();
    fclose(fp);
    fp = fopen("src/game.h", "w");
    if(fp == NULL) W_ERROR();
    write_copyright(fp, author_name, argument, year);
    if(append_file(fp, shared_dir, "basefile.h") == 0) W_ERROR();
    fclose(fp);
    fp = fopen("src/includes.h", "w");
    write_copyright(fp, author_name, argument, year);
    fprintf(fp, "\n#include \"weaver/weaver.h\"\n");
    fprintf(fp, "\n#include \"game.h\"\n");
    fclose(fp);
  }
  else{
    fprintf(stderr, "ERROR: %s is not a valid project name.", argument);
    return_value = 1;
  }
  END();
}
```

7.7. Criar novo plugin

Este aso de uso invocado quando temos dois argumentos, o primeiro `"--plugin"` e o segundo o nome de um novo plugin, o qual deve ser um nome nico, sem conflitar com qualquer outro dentro de `plugins/`. Devemos estar em um diretrio Weaver para fazer isso.

Seo: Caso de uso 7: Criar novo plugin:

```
if(inside_weaver_directory && have_arg && !strcmp(argument, "--plugin") &&
   arg_is_valid_plugin){
  char *buffer;
  FILE *fp;
  /* Criando o arquivo: */
  buffer = concatenate("plugins/", argument2, ".c", "");
  if(buffer == NULL) W_ERROR();
  fp = fopen(buffer, "w");
  if(fp == NULL) W_ERROR();
  write_copyright(fp, author_name, project_name, year);
  fprintf(fp, "#include \"../src/weaver/weaver.h\"\n\n");
  fprintf(fp, "void _init_plugin_%s(W_PLUGIN){\n\n}\n\n", argument2);
  fprintf(fp, "void _fini_plugin_%s(W_PLUGIN){\n\n}\n\n", argument2);
  fprintf(fp, "void _run_plugin_%s(W_PLUGIN){\n\n}\n\n", argument2);
```

```
  fprintf(fp, "void _enable_plugin_%s(W_PLUGIN){\n\n}\n\n", argument2);
  fprintf(fp, "void _disable_plugin_%s(W_PLUGIN){\n\n}\n", argument2);
  fclose(fp);
  free(buffer);
  END();
}
```

7.8. Criar novo shader

Este caso de uso similar ao anterior, mas possui algumas diferenas. Todo shader ser um novo arquivo no formato GLSL dentro do diretrio `shaders`. E alm disso, seu nome ter sempre o formato dado pela expresso regular `[0-9][0-9]*-.*`. O(s) dgito(s) na primeira parte do nome deve ser nico para cada shader de um mesmo projeto. E os nmeros representados por tais dgitos devem ser sempre sequenciais, comeando no 1 e incrementando-o a cada novo shader.

Este caso de uso ser invocado somente quando o nosso primeiro argumento for ``--shader'' e o segundo for um nome qualquer. No precisamos realmente forar uma restrio nos nomes dos shaders, pois sua conveno numrica garante que cada um ter um nome nico e no-conflitante.

Para garantir isso, o cdigo dever contar quantos arquivos com extenso GLSL existem no diretrio dos shaders e criar um novo shader com nome `DD-XX.glsl`, onde `DD` o nmero de arquivos que existia mais 1 e `XX` o nome escolhido passado como segundo argumento para o programa. Mas se existirem lacunas na numerao de shaders, por exemplo existir um shader 1 e um 3 sem existir o 2, daremos preferncia para cobrir a lacuna. O contedo base de um shader ser obtido de um arquivo onde o programa Weaver est instalado.

Depois de descobrir a numerao do novo shader, basta criarmos ele como um arquivo vazio e depois copiarmos o contedo de um modelo j existente em nosso diretrio de instalao.

O cdigo deste caso de uso ento:

Seo: Caso de uso 8: Criar novo shader:

```
if(inside_weaver_directory && have_arg && !strcmp(argument, "--shader") &&
   argument2 != NULL){
    FILE *fp;
    size_t tmp_size, number = 0;
    int shader_number;
    char *buffer;
     <Seo a ser Inserida: Shader: Conta nmero de arquivos e obtm nmero do shader>
    // Criando o shader:
    tmp_size = number / 10 + 7 + strlen(argument2);
    buffer = (char *) malloc(tmp_size);
    if(buffer == NULL) W_ERROR();
    buffer[0] = '\0';
    snprintf(buffer, tmp_size, "%d-%s.glsl", (int) number, argument2);
    fp = fopen(buffer, "w");
    if(fp == NULL){
        free(buffer);
        W_ERROR();
    }
    if(append_file(fp, shared_dir, "shader.glsl") == 0) W_ERROR();
    fclose(fp);
    free(buffer);
    END();
}
```

A parte de contar onter o nmero do novo shader ocorre de maneira diferente no Unix e no Windows devido API diferente para lidar com o sistema de arquivos. Tirando as

particularidades de como iterar sobre arquivos em um diretrio, o que faremos iterar em cada arquivo no diretrio `shaders` de nosso projeto que no seja um diretrio, tenha extenso GLSL e tenha seu nome comeado com um nmero positivo. Chamaremos tal nmero de `number`. Teremos um vetor booleano inicialmente marcado inteiramente como falso. Ao chegar em cada um destes arquivos, marcamos no vetor booleano a informao de que o shader de nmero `number` existe colocando o valor verdadeiro na posio do vetor reservada para ele. Depois de iterarmos sobre cada um dos arquivos, acharemos a primeira posio do vetor que ainda est marcada como falsa. Sua posio indica qual nmero de shadr ainda no foi usado e o nmero que escolheremos.

Complexidades adicionais neste cdigo envolvem apenas tomarmos o cuidado para que o nosso vetor booleano sempre tenha um tamanho adequado. Para isso tentamos alocar ele inicialmente com 128 espaos, mas se acharmos shaders com nmeros altos o bastante, o realocaremos para lidar com o nmero maior.

O cdigo para isso no Linux ser:

Seo: Shader: Conta nmero de arquivos e obtm nmero do shader:

```
#if !defined(_WIN32)
{
  size_t i, max_number = 0;
  DIR *shader_dir;
  struct dirent *dp;
  char *p;
  bool *exists;
  size_t exists_size = 128;
  shader_dir = opendir("shaders/");
  if(shader_dir == NULL)
    W_ERROR();
  exists = (bool *) malloc(sizeof(bool) * exists_size);
  if(exists == NULL){
    closedir(shader_dir);
    W_ERROR();
  }
  for(i = 0; i < exists_size; i ++)
    exists[i] = false;
  while((dp = readdir(shader_dir)) != NULL){
    if(dp -> d_name == NULL) continue;
    if(dp -> d_name[0] == '.') continue;
    if(dp -> d_name[0] == '\0') continue;
    buffer = concatenate("shaders/", dp -> d_name, "");
    if(buffer == NULL) W_ERROR();
    if(directory_exist(buffer) != EXISTE_E_EH_ARQUIVO){
      free(buffer);
      continue;
    }
    for(p = buffer; *p != '\0'; p ++);
    p -= 5;
    if(strcmp(p, ".glsl") && strcmp(p, ".GLSL")){
      free(buffer);
      continue;
    }
    number = atoi(buffer);
    if(number == 0){
```

```
      free(buffer);
      continue;
    }
    if(number > max_number)
      max_number = number;
    if(number > exists_size){
      if(number > exists_size * 2)
        exists_size = number;
      else
        exists_size *= 2;
      exists = (bool *) realloc(exists, exists_size * sizeof(bool));
      if(exists == NULL){
        free(buffer);
        closedir(shader_dir);
        W_ERROR();
      }
      for(i = exists_size / 2; i < exists_size; i ++)
        exists[i] = false;
    }
    exists[number - 1] = true;
    free(buffer);
  }
  closedir(shader_dir);
  for(i = 0; i <= max_number; i ++)
    if(exists[i] == false){
      shader_number = i + 1;
      break;
    }
  free(exists);
}
#endif
```

No Windows, o cdigo para iterar sobre arquivos diferente, mas o restante no muda:

Seo: Shader: Conta nmero de arquivos e obtm nmero do shader:

```
#if defined(_WIN32)
{
  int i;
  char *p;
  bool *exists;
  size_t exists_size = 128;
  int number, max_number = 0;
  WIN32_FIND_DATA file;
  HANDLE shader_dir = NULL;
  shader_dir = FindFirstFile("shaders\\", &file);
  if(shader_dir == INVALID_HANDLE_VALUE)
    W_ERROR();
  exists = (bool *) malloc(sizeof(bool) * exists_size);
  if(exists == NULL){
    FindClose(shader_dir);
    W_ERROR();
  }
```

```
  for(i = 0; i < exists_size; i ++)
    exists[i] = false;
  do{
    if(file.cFileName == NULL) continue;
    if(file.cFileName[0] == '.') continue;
    if(file.cFileName[0] == '\0') continue;
    buffer = concatenate("shaders\\", file.cFileName, "");
    if(buffer == NULL) W_ERROR();
    if(directory_exist(buffer) != EXISTE_E_EH_ARQUIVO){
      free(buffer);
      continue;
    }
    for(p = buffer; *p != '\0'; p ++);
    p -= 5;
    if(strcmp(p, ".glsl") && strcmp(p, ".GLSL")){
      free(buffer);
      continue;
    }
    number = atoi(buffer);
    if(number == 0){
      free(buffer);
      continue;
    }
    if(number > max_number)
      max_number = number;
    if(number > exists_size){
      if(number > exists_size * 2)
        exists_size = number;
      else
        exists_size *= 2;
      exists = (bool *) realloc(exists, exists_size * sizeof(bool));
      if(exists == NULL){
        free(buffer);
        FindClose(shader_dir);
        W_ERROR();
      }
      for(i = exists_size / 2; i < exists_size; i ++)
        exists[i] = false;
    }
    exists[number - 1] = true;
    free(buffer);
  }while(FindNextFile(shader_dir, &file) != 0);
  FindClose(shader_dir);
  for(i = 0; i <= max_number; i ++)
  if(exists[i] == false){
    shader_number = i + 1;
    break;
  }
  free(exists);
}
```

```
#endif
```

7.9. Criar novo loop principal

Este caso de uso ocorre quando o segundo argumento `--loop` e quando o prximo argumento for um nome vlido para uma funo. Se no for, imprimimos uma mensagem de erro para avisar.

Neste caso no podemos apenas copiar o contedo de um arquivo base para formar o arquivo com um novo mdulo do projeto Weaver, pois esse novo arquivo ter definida uma funo com um nome fornecido pelo usurio. Ento apenas criamos e preenchemos o arquivo na hora com contedo defiido no prprio cdigo abaixo.

Seo: Caso de uso 9: Criar novo loop principal:

```
if(inside_weaver_directory && !strcmp(argument, "--loop")){
  if(!arg_is_valid_function){
    if(argument2 == NULL)
      fprintf(stderr,
              "ERROR: You should pass a name for your new loop.\n");
    else
      fprintf(stderr, "ERROR: %s not a valid loop name.\n", argument2);
    W_ERROR();
  }
  char *filename;
  FILE *fp;
  // Criando LOOP_NAME.c
  filename = concatenate(project_path, "src/", argument2, ".c", "");
  if(filename == NULL) W_ERROR();
  fp = fopen(filename, "w");
  if(fp == NULL){
    free(filename);
    W_ERROR();
  }
  write_copyright(fp, author_name, project_name, year);
  fprintf(fp, "#include \"%s.h\"\n\n", argument2);
  fprintf(fp, "MAIN_LOOP %s(void){\n", argument2);
  fprintf(fp, " LOOP_INIT:\n\n");
  fprintf(fp, " LOOP_BODY:\n");
  fprintf(fp, "  if(W.keyboard[W_ANY])\n");
  fprintf(fp, "    Wexit_loop();\n");
  fprintf(fp, " LOOP_END:\n");
  fprintf(fp, "  return;\n");
  fprintf(fp, "}\n");
  fclose(fp);
  // Criando LOOP_NAME.h
  filename[strlen(filename)-1] = 'h';
  fp = fopen(filename, "w");
  if(fp == NULL){
    free(filename);
    W_ERROR();
  }
  write_copyright(fp, author_name, project_name, year);
  fprintf(fp, "#ifndef _%s_h_\n", argument2);
```

```
  fprintf(fp, "#define _%s_h_\n#include \"weaver/weaver.h\"\n\n", argument2);
  fprintf(fp, "#include \"includes.h\"\n\n");
  fprintf(fp, "MAIN_LOOP %s(void);\n\n", argument2);
  fprintf(fp, "#endif\n");
  fclose(fp);
  free(filename);
  // Atualizando src/includes.h
  fp = fopen("src/includes.h", "a");
  fprintf(fp, "#include \"%s.h\"\n", argument2);
  fclose(fp);
}
```

8. Concluso

Isso finaliza todo o cdigo necessrio para que o programa Weaver possa gerenciar projetos de jogos feitos com o motor Weaver.

O programa apresentado aqui ainda no representa todo o gerenciamento de um projeto. Uma parte no retratada aqui um instalador que em sistemas Unix representado por um `Makefile` responsvel por instalar o motor Weaver no local adequado e em sistemas Windows tem a forma de um pacote MSIX.

Alm disso, o cdigo das bibliotecas em si tambm fazem parte do motor Weaver, mas tero o seu cdigo descrito em outros artigos.

Alguns cdigos como o cdigo-base para shaders e novos projetos podem ser encontrados junto com o cdigo-fonte de Weaver, no diretrio `project`.

Por fim, em sistemas Unix h um Makefile para cada projeto, que tambm realiza muito do desenvolvimento. No Windows, se utiliza-se o Visual Studio ao invs de ferramentas Unix, esta parte do gerenciamento ser feita por um conjunto de regras.