Métodos de Programação II

Trabalho Prático 3

Leandro Alves A82157
J. Eduardo Santos A82350
David Machado A82381

Junho de 2017

Conteúdo

1 Introdução

Este relatório tem como objetivo descrever o trabalho desenvolvido no âmbito da Unidade Curricular Métodos de Programação II, do Mestrado Integrado em Engenharia de Telecomunicações e Informática da Universidade do Minho.

Este trabalho foi desenvolvido através da linguagem de programação C com o auxílio a estruturas, ficheiros sequênciais, listas ligadas, apontadores, memória alocada dinamicamente e listas generalizadas, visando o desenvolvimento de um jogo de perguntas e respostas como foi proposto no enunciado.

2 Problema

O objetivo deste trabalho prático é fazer um jogo de perguntas e respostas através de listas ligadas.

O nosso jogo chama-se *The QUIZZER by EdKeriohn* e o objetivo do jogador é de derrotar personagens famosas respondento acertadamente a perguntas de categorias relacionadas com essas personagens.

Neste programa, as perguntas e respetivas opções (certas e erradas) devem ser carregadas através dum ficheiro de texto construído previamente à mão, sendo armazenadas em estruturas e guardadas em memória através de listas generalizadas por ordem de dificuldade.

Como neste jogo optamos por deixar as perguntas com o mesmo grau de dificuldade, em vez de guarda-las por essa ordem, estas são guardadas por ordem alfabética.

Durante o jogo, este deve decorrer enquanto forem apresentadas perguntas para o jogador responder, sendo validadas as suas respostas.

O jogo deverá terminar quando o jogador perder ou ganhar.

Também é necessário o armazenamento dos dados num ficheiro para que se possa sair do programa sem perder os mesmos.

3 Jogabilidade

Este jogo é baseado nos estilos de combate estratégico e de luta onde o objetivo é derrotar o adversário impedindo de ser atacado por este.

O jogador e cada personagem têm pontos de vida que são reduzidos se forem atacados, o jogador ataca se acertar uma pergunta e se errar é atacado.

Se uma personagem perder todos os pontos de vida, esta é derrotada, se for o jogador a perde-los, este perde o jogo.

As perguntas são classificadas por categorias que poderão ser: Arte, Entretenimento, Geografia, História, Ciência e Desporto.

O dano que o jogador ou o oponente provocam é calculado através dos pontos de ataque de cada um vezes um número aleatório entre 0.85 e 1.

Há 10% de probabilidade de cada golpe ser crítico, provocando o dobro do dano se assim for.

Há também 20% de probabilidade do ataque falhar, tornando o dano nulo.

3.1 Modos de jogo

Este jogo apresenta três modos de jogo: *Elite Six* (campanha), *Survival* (sobrevivência) e *Hardcore*.

No modo ***Elite Six***, o objetivo é de derrotar seis personagens famosas que correspondem às seis categorias do jogo. Em cada nível, o jogador pode escolher a personagem que quiser enfrentar mas nunca pode escolher uma categoria repetida. Estas personagens ganham mais pontos de vida e de ataque mas o jogador pode desbloquear um *power-up* a cada nível que o beneficiará. Além disso, em cada pergunta, o jogador poderá usar uma habilidade para lhe ser mais fácil achar a reposta certa. Se o jogador conseguir derrotar a *Elite Six*, este enfrentará o Campeão de Elite, especialista em todas as categorias.

No modo ***Survival*** o objetivo é aguentar o maior número de rondas possível sem perder. Neste modo de jogo, a categoria de cada pergunta é aleatória e basta acertar uma pergunta para passar para a próxima ronda. O jogador pode ainda usufruir de alguns *power-ups* inicialmente desbloqueados mas não pode usar habilidades.

O modo ***Hardcore*** é idêntico ao *Elite Six* mas não é permitido ao jogador usar *power-ups* nem abilidades.

3.2 *Power-ups*

Nos modos de jogo *Elite Six* e *Survival*, o jogador poderá usufruir de *power-ups* que benificiam o jogador de diversas maneiras.

No modo *Elite Six*, antes de enfrentar o adversário, este poderá escolher um *power-up* que não poderá ser escolhido posteriormente mas nunca o irá perder, sendo cada *power-up* acumulável.

No modo *Survival*, o jogador começa o jogo com alguns *power-ups* predefinidamente desbloqueados e não poderá desbloquear mais nenhum.

Tipos de *power-ups*:

Ability Up **-** O jogador pode usar cada habilidade mais uma vez.

Precision **-** Os ataques do jogador nunca falham.

Recover **-** Sempre que o jogador derrotar um adversário, são-lhe acrescentados 50 pontos de vida.

Regeneration **-** São adicionados 5 pontos de vida a cada pergunta respondida.

Resistance **-** O utilizador recebe apenas 33% do dano infligido.

Strength **-** Aumenta o ataque do utilizador em 50%.

Vampire **-** O dano retirado é convertido em pontos de vida adicionais.

3.3 Habilidades

No modo de jogo *Elite Six*, o jogador poderá usufruir de habilidades antes de responder a cada pergunta que lhe aumentarão a chance de responder acertadamente à mesma sendo estas esgotadas depois de serem usadas.

Tipos de habilidades:

Bomb **-** O jogador é informado acerca de duas opções que estão erradas.

Double Chance **-** Se o jogador responder incorretamente ao uma pergunta, este poderá responder mais uma vês.

Skip **-** A pergunta é trocada por uma da mesma categoria.

Summon Ally **-** O jogador é informado com 70% de eficácia acerca da reposta certa.

4 Resolução

4.1 Inicialização do Código

1. Definição do tipo de 5 estruturas para as estatísticas (nome, respostas corretas de cada categoria, adversários derrotados (elite 6, campeão e *minions*), record do nível de cada modo de jogo e máximo de repostas corretas seguidas), *powerups* (nome, abreviação, descrição e uso), habilidades (nome, descrição e uso), membros da elite (categoria, 6 nomes e uso) e *minions* (categoria e 10 nomes), sendo estas 4 últimas guardadas num array cada um pois há várias entidades para estas estruturas.

2. Definição do tipo e do apontador de uma estrutura para as opções erradas das perguntas com cada opção errada e a declaração recursiva da mesma estrutura para a próxima opção.

3. Definição do tipo e do apontador de uma estrutura para as perguntas com a respetiva pergunta, a resposta correta e as declarações recursivas para as próximas perguntas e a declaração das opções erradas vindas da estrutura anterior.

4.2 Função *main*

1. As 5 primeiras estruturas (*stat*, *pwrup*, *abil*, *elite* e *minion*) são declaradas, sendo as 4 últimas vetores.

2. São chamadas as funções que irão ler os ficheiros correspondentes às estruturas referidas que armazenaram os seus dados nas mesmas (*Stats_Read*, *Pwrups_Read*, *Abils_Read*, *Elite_Read* e *Minions_Read*).

3. Enquanto o utilizador não introduzir o número correspondente à saída do programa, o menu aparece com as opções para jogar, ver as estatísticas, os créditos e a saída do programa e a opção do utilizador irá direciona-lo à respetiva função.

4. Se o utilizador optar por sair do programa, as estatísticas alteradas serão guardadas no ficheiro correspondente pois é chamada uma função para tal (*Stats_Write*).

4.3 Funções de leitura

As funções *Stats_Read*, *Pwrups_Read*, *Abils_Read*, *Elite_Read* e *Minions_Read* servem para ler os ficheiros correspondentes às mesmas, armazenando os dados dos mesmos em estruturas próprias.

4.4 Função *New_Game*

Enquanto o utilizador não quiser sair do menu dos modos de jogo, este poderá escolher um dos 3 modos disponíveis: *Elite Six*, *Survival* e *Hardcore*. Todas as opções chamam a mesma função mas com o número escolhido como argumento.

4.5 Função *Play*

1. A variável *play* assume o o número introduzido na função anterior referente ao modo de jogo escolhido e como cada modo tem as suas diferenças, há certas condições que só ocorrem dependendo do valor dessa variável (habilidades usadas, uso de *powerups*, etc.).

2. Enquanto o jogo não terminar (*finish* = 0), este irá decorrer até *finish* tomar o valor de 1 que acontece quando os pontos de vida do jogador chegarem a 0 ou quando este conseguir derrotar o campeão nos modos de jogo *Elite Six* e *Hardcore*.

3. Há medida que o jogo decorre, são chamadas várias funções (*Status*, *Elite*, *Powerups* e *Questions*).

4. O fundamental desta função já foi referido anteriormente na jogabilidade.

4.6 Função *Status*

Nesta função são apresentados os dados e características do jogo, do jogador e do oponente (nome, categoria, pontos de vida, ataque, *powerups*, modo de jogo, nível/ronda e perguntas corretas consecutivas).

4.7 Função *Elite*

1. A lista dos adversários disponíveis e categorias derrotadas é apresentada e o utilizador introduzirá o número do adversário que quiser enfrentar.

2. Esta função só aparece no modo *Elite Six*.

4.8 Função *Powerups*

1. A lista dos *powerups* disponíveis e já em uso é apresentada e o utilizador introduzirá o número do adversário que quiser enfrentar.

2. Esta função só aparece no modo *Elite Six*.

4.9 Função *Questions*

1. O apontador da estrutura das perguntas é declarado nulo.

2. A pergunta é selecionada aleatoriamente através da função *Quests_Read* que recebe como argumento a categoria escolhida na função *Play*.

3. A pergunta é apresentada no ecrã e o número da opção correta é retornado pela função *Options*.

4. O utilizador introduz um número de uma das 4 opções disponíveis ou da habilidade que pretender usar e se este corresponder a uma habilidade, a função correspondente (*Bomb*, *Double_Chance*, *Skip* e *Summon_Ally*) é chamada retornando o número da opção que este escolher nessas funções.

5. A função retorna 1 se a opção for correta, 2 se for errada e 0 se o utilizador quiser desistir.

4.10 Função *Quests_Read*

1. Os apontadores *head* e *new* da estrutura das perguntas e *wrong_new* e *wrong_head* da estrutura das opções erradas são declarados nulos.

2. O ficheiro das perguntas é aberto em modo de leitura.

3. Enquanto o programa estiver a ler as perguntas, se o número da categoria escolhida for o número da categoria, é alocado um espaço de memória do tamanho da estrutura e o seu endereço é guardado na variável *new* e as perguntas e a respota correta são lidas.

4. É alocado um espaço de memória do tamanho da estrutura das opções erradas e o seu endereço é guardado na variável *wrong_new* para cada opção ser lida.

5. A variável *head* toma o valor do retorno da função *Insert*, depois o número de perguntas é incrementado.

6. Se for lido um número de outra categoria, o programa passa as perguntas e as opções à frente até achar o número da categoria escolhida.

7. A pergunta que vai ser apresentada será uma aleatória entre as lidas pois o programa irá percorrer a lista das perguntas até encontrar esse número.

4.11 Função *Insert*

1. O nível de dificuldade das perguntas depende somente da sabedoria do jogador portanto não organizamos dessa maneira. Em vez disso, decidimos optar por ordenar as perguntas alfabeticamente.

2. Enquanto a lista das perguntas estiver a ser percorrida, o programa irá verificar pergunta a pergunta e, caso a primeira seja maior do que a segunda, a funçao é chamada recursivamente para ser verifica a próxima pergunta.

3. Se estiver por ordem alfabética inversa, a variável *head* será a próxima pergunta.

4.12 Função *Options*

1. O apontador *wrong* da estrutura das opções erradas é declarado.

2. Nesta função, o programa irá distribuir aleatoriamente as 4 opções da pergunta.

3. Se o utilizador puder usar habilidades, estas serão-lhe apresentadas.

4. Esta função retorna o número da opção que corresponde à correta.

4.13 Funções de habilidades

4.13.1 Função *Bomb*

O programa irá guardar num vetor duas opções erradas geradas aleatoriamente que serão apresentadas ao utilizador e este escolherá a opção que achar correta.

4.13.2 Função *Double_Chance*

O utilizador introduzirá uma opção e se esta estiver errada, poderá introduzir outra novamente.

4.13.3 Função *Skip*

O programa irá procurar outra pergunta da categoria que está em jogo e chamará as funções *Quests_Read* e *Options* para ser apresentada a novas pergunta e as opções para ser respondida.

4.13.4 Função *Summon_Ally*

1. O programa tem 70% de apresentar a resposta correta, nos 30% será apresentada uma das 3 erradas e o utilizador introduzirá a opção que pretender.

2. Todas estas funções retornam 1 se a opção for correta e 2 se forem erradas.

4.14 Função *Statistics*

1. As estatísticas do progresso do utilizador são apresentadas.

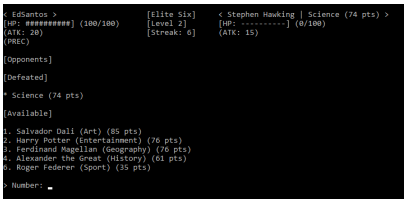
2. O utilizador poderá alterar o nome e apagar o progresso (funções *Name* e *Delete*) nesta função.

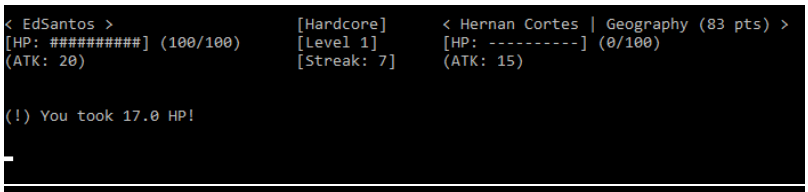
4.15 Função *Credits*

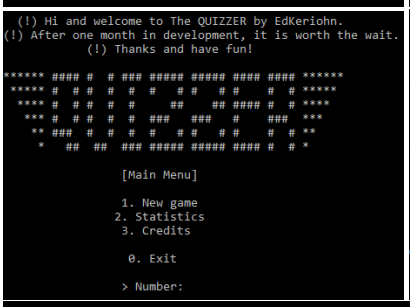
Os créditos do jogo são apresentados.

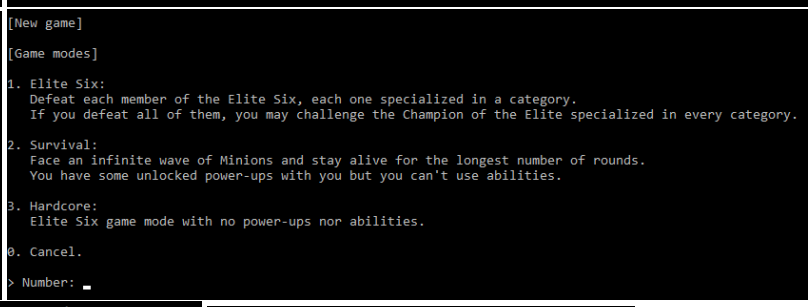
5 Exemplo

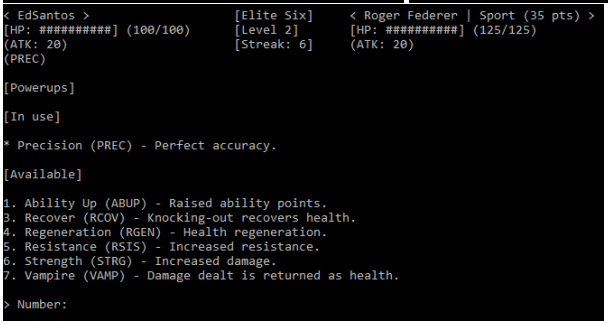
5.1 Capturas de ecrã

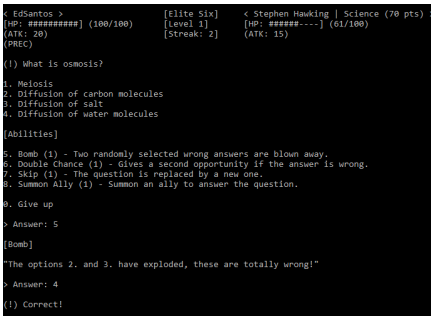


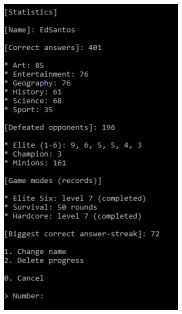












5.2 *Gameplay*

- ***Elite Six*** **-** https://youtu.be/djG5STzFkOQ
- ***Survival*** **-** https://youtu.be/g6_N_XRKgoQ

6 Implementação

6.1 Código em *C*

```c
//  mp216TP3Gr06.c
//  TP3 | Jogo Quiz

//  Grupo 6 - PL1
//  Leandro Alves        | A82157
//  J. Eduardo Santos    | A82350
//  David Machado        | A82381

//  Versao final 1.0

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#define MAX 150

typedef struct stat_s
{
    char name [MAX];
    int correct [6];
    int defeated [8];
    int lv [3];
    int streak;
}   stat_t;

typedef struct pwrup_s
{
    char name [MAX];
    char abrev [MAX];
    char desc [MAX];
    float use;
}   pwrup_t;

typedef struct abil_s
{
    char name [MAX];
    char desc [MAX];
    int use;
}   abil_t;

typedef struct elite_s
{
    char cat [MAX];
    char name [6] [MAX];
    int use;
}   elite_t;

typedef struct minion_s
{
    char cat [MAX];
    char name [10] [MAX];
}   minion_t;
```

```
54
55 typedef struct wrong_s
56 {
57     char option [MAX];
58     struct wrong_s *next;
59 }   wrong_t, *wrong_p;
60
61 typedef struct quest_s
62 {
63     char question [MAX];
64     char correct [MAX];
65     struct quest_s *next;
66     struct wrong_s *answer;
67 }   quest_t, *quest_p;
68
69 void Stats_Read (stat_t *stat)
70 {
71     int i = 0;
72     char temp [MAX];
73
74     FILE *file;
75
76     file = fopen ("statistics.txt", "r");
77
78     if (file)
79     {
80         fgets (stat -> name, MAX, file);
81         stat -> name [strlen (stat -> name) - 1] = '\0';
82
83         for (i = 0; i < 6; i ++)
84         {
85             fgets (temp, MAX, file);
86             stat -> correct [i] = atoi (temp);
87         }
88         for (i = 0; i < 8; i ++)
89         {
90             fgets (temp, MAX, file);
91             stat -> defeated [i] = atoi (temp);
92         }
93         for (i = 0; i < 3; i ++)
94         {
95             fgets (temp, MAX, file);
96             stat -> lv [i] = atoi (temp);
97         }
98         fgets (temp, MAX, file);
99         stat -> streak = atoi (temp);
100
101         fclose (file);
102     }
103 }
104 void Pwrups_Read (pwrup_t *pwrup)
105 {
106     int i = 0;
107     char temp [MAX];
108
109     FILE *file;
110
111     file = fopen ("powerups.txt", "r");
```

```
112
113     if (file)
114     {
115         while (!feof (file))
116         {
117             fgets (pwrup[i].name, MAX, file);
118             pwrup[i].name [strlen (pwrup[i].name) - 1] = '\0';
119
120             fgets (pwrup[i].abrev, MAX, file);
121             pwrup[i].abrev [strlen (pwrup[i].abrev) - 1] = '\0';
122
123             fgets (pwrup[i].desc, MAX, file);
124             pwrup[i].desc [strlen (pwrup[i].desc) - 1] = '\0';
125
126             fgets (temp, MAX, file);
127             pwrup[i].use = atoi (temp);
128
129             i ++;
130         }
131         fclose (file);
132     }
133 }
134 void Abils_Read (abil_t *abil)
135 {
136     int i = 0;
137     char temp [MAX];
138
139     FILE *file;
140
141     file = fopen ("abilities.txt", "r");
142
143     if (file)
144     {
145         while (!feof (file))
146         {
147             fgets (abil[i].name, MAX, file);
148             abil[i].name [strlen (abil[i].name) - 1] = '\0';
149
150             fgets (abil[i].desc, MAX, file);
151             abil[i].desc [strlen (abil[i].desc) - 1] = '\0';
152
153             fgets (temp, MAX, file);
154             abil[i].use = atoi (temp);
155
156             i ++;
157         }
158         fclose (file);
159     }
160 }
161 void Elite_Read (elite_t *elite)
162 {
163     int i = 0, j;
164     char temp [MAX];
165
166     FILE *file;
167
168     file = fopen ("elite.txt", "r");
169
```

```
170     if (file)
171     {
172         while (!feof (file))
173         {
174             fgets (elite[i].cat, MAX, file);
175             elite[i].cat [strlen (elite[i].cat) - 1] = '\0';
176 
177             for (j = 0; j < 6; j ++)
178             {
179                 fgets (elite[i].name[j], MAX, file);
180                 elite[i].name[j] [strlen (elite[i].name[j]) - 1] = '\0';
181             }
182             fgets (temp, MAX, file);
183             elite[i].use = atoi (temp);
184 
185             i ++;
186         }
187         fclose (file);
188     }
189 }
190 void Minions_Read (minion_t *minion)
191 {
192     int i = 0, j;
193     char temp [MAX];
194 
195     FILE *file;
196 
197     file = fopen ("minions.txt", "r");
198 
199     if (file)
200     {
201         while (!feof (file))
202         {
203             fgets (minion[i].cat, MAX, file);
204             minion[i].cat [strlen (minion[i].cat) - 1] = '\0';
205 
206             for (j = 0; j < 10; j ++)
207             {
208                 fgets (minion[i].name[j], MAX, file);
209                 minion[i].name[j] [strlen (minion[i].name[j]) - 1] = '\0';
210             }
211             i ++;
212         }
213         fclose (file);
214     }
215 }
216 quest_p Insert (quest_p head, quest_p new)
217 {
218     if (head)
219     {
220         if (strcmp (head -> question, new -> question) > 0)
221         {
222             head -> next = Insert (head -> next, new);
223 
224         }
225         else
226         {
227             new -> next = head;
```

```
228             head = new;
229         }
230     }
231     else
232     {
233        head = new;
234     }
235     return head;
236 }
237 quest_p Quests_Read (int cat)
238 {
239     int temp_cat, i = 0, count = 0, selected;
240     char temp [MAX], question [MAX], answer [MAX];
241 
242     quest_p head = NULL, new = NULL;
243     wrong_p wrong_new = NULL, wrong_head = NULL;
244 
245     FILE *file;
246 
247     file = fopen ("questions.txt", "r");
248 
249     if (file)
250     {
251         while (fgets (temp, MAX, file))
252         {
253             temp_cat = atoi (temp);
254 
255             if (temp_cat == cat)
256             {
257                 new = (quest_p) malloc (sizeof (quest_t));
258 
259                 fgets (new -> question, MAX, file);
260                 fgets (new -> correct, MAX, file);
261 
262                 for (i = 0; i < 3; i ++)
263                 {
264                     wrong_new = (wrong_p) malloc (sizeof (wrong_t));
265 
266                     fgets (wrong_new -> option, MAX, file);
267                     wrong_new -> next = wrong_head;
268                     wrong_head = wrong_new;
269                 }
270                 new -> answer = wrong_new;
271 
272                 head = Insert (head, new);
273 
274                 count ++;
275             }
276             else
277             {
278                 for (i = 0; i < 5; i ++)
279                 {
280                     fgets (temp, MAX, file);
281                 }
282             }
283         }
284         selected = rand () %count;
285 
```

```
286         i = 0;
287
288         while (selected != i)
289         {
290             head = head -> next;
291             i ++;
292         }
293     }
294     fclose (file);
295
296     return head;
297 }
298 void Stats_Write (stat_t *stat)
299 {
300     int i;
301
302     FILE *file;
303
304     file = fopen ("statistics.txt", "w");
305
306     fprintf (file, "%s\n", stat -> name);
307
308     for (i = 0; i < 6; i ++)
309     {
310         fprintf (file, "%d\n", stat -> correct [i]);
311     }
312     for (i = 0; i < 8; i ++)
313     {
314         fprintf (file, "%d\n", stat -> defeated [i]);
315     }
316     for (i = 0; i < 3; i ++)
317     {
318         fprintf (file, "%d\n", stat -> lv [i]);
319     }
320     fprintf (file, "%d\n", stat -> streak);
321
322     fclose (file);
323 }
324 int Powerups (int error, pwrup_t *pwrup, abil_t *abil)
325 {
326     int i, used_pwrups = 0, avail_pwrups = 0, n;
327     char temp [MAX];
328
329     printf ("[Powerups]\n");
330
331     for (i = 0; i < 7; i ++)
332     {
333         if (pwrup[i].use == 1)
334         {
335             used_pwrups ++;
336         }
337         else
338         {
339             avail_pwrups ++;
340         }
341     }
342     if (used_pwrups > 0)
343     {
```

```
344         if (pwrup[0].use == 0)
345         {
346             printf ("\n[In use]\n\n");
347         }
348         else
349         {
350             if (used_pwrups > 1)
351             {
352                 printf ("\n[In use / already used]\n\n");
353             }
354             else
355             {
356                 printf ("\n[Already used]\n\n");
357             }
358         }
359         for (i = 0; i < 7; i ++)
360         {
361             if (pwrup[i].use == 1)
362             {
363                 printf ("* %s (%s) - %s\n", pwrup[i].name, pwrup[i].abrev, pwrup[i].
                        desc);
364             }
365         }
366     }
367     if (avail_pwrups > 0)
368     {
369         printf ("\n[Available]\n\n");
370 
371         for (i = 0; i < 7; i ++)
372         {
373             if (pwrup[i].use == 0)
374             {
375                 printf ("%d. %s (%s) - %s\n", i + 1, pwrup[i].name, pwrup[i].abrev,
                        pwrup[i].desc);
376             }
377         }
378     }
379     if (error == 0)
380     {
381         printf ("\n> Number: ");
382     }
383     else
384     {
385         printf ("\n> Please, choose a valid number: ");
386     }
387     fgets (temp, sizeof (temp), stdin);
388     n = atoi (temp);
389 
390     if (n == 1 && pwrup[n - 1].use == 0)
391     {
392         for (i = 0; i < 4; i ++)
393         {
394             abil[i].use ++;
395         }
396     }
397     pwrup[n - 1].use ++;
398 
399     if (n < 1 || n > 7 || pwrup[n - 1].use == 2)
```

```
400     {
401         error = 1;
402 
403         if (pwrup[n - 1].use == 2)
404         {
405           pwrup[n - 1].use = 1;
406         }
407     }
408     else
409     {
410         error = 0;
411     }
412     return error;
413 }
414 int Options (int using_abil, int play, abil_t *abil, quest_p quest)
415 {
416     int i, j, correct, k, used_abils = 0, array [4] = {0, 0, 0, 0};
417 
418     wrong_p wrong;
419 
420     for (i = 1; i <= 4; i ++)
421     {
422         j = rand () %4;
423         while (array [j] == 1)
424         {
425             j = rand () %4;
426         }
427         array [j] = 1;
428         if (j == 0)
429         {
430             printf ("%d. %s", i, quest -> correct);
431             correct = i;
432         }
433         else
434         {
435             k = 1;
436             wrong = quest -> answer;
437             while (k < j)
438             {
439                 wrong = wrong -> next;
440                 k ++;
441             }
442             printf ("%d. %s", i, wrong -> option);
443         }
444     }
445     for (i = 0; i < 4; i ++)
446     {
447         if (abil[i].use > 0)
448         {
449             used_abils ++;
450         }
451     }
452     if (used_abils > 0 && using_abil == 0 && play == 1)
453     {
454         printf ("\n[Abilities]\n\n");
455 
456         for (i = 0; i < 4; i ++)
457         {
```

```
458             if (abil[i].use > 0)
459             {
460                 printf ("%d. %s (%d) - %s\n", i + 5, abil[i].name, abil[i].use, abil
                        [i].desc);
461             }
462         }
463     }
464     if (using_abil == 0)
465     {
466         printf ("\n0. Give up\n");
467     }
468     return correct;
469 }
470 int Bomb (int correct, abil_t *abil)
471 {
472     int i, answer, bomb [2];
473     char temp [MAX];
474 
475     for (i = 0; i < 2; i ++)
476     {
477         bomb [i] = (rand () %3) + 1;
478 
479         while (bomb [i] == correct || bomb [0] == bomb [1])
480         {
481             bomb [i] = (rand () %3) + 1;
482         }
483     }
484     printf ("\n[Bomb]\n\n\"The options %d. and %d. have exploded, these are totally
            wrong!\"\n\n", bomb [0], bomb [1]);
485 
486     printf("> Answer: ");
487     fgets (temp, MAX, stdin);
488     answer = atoi (temp);
489 
490     if (answer == correct)
491     {
492         correct = 1;
493     }
494     else
495     {
496         correct = 2;
497     }
498     return correct;
499 }
500 int Double_Chance (int correct, abil_t *abil)
501 {
502     int stop = 0, i = 0, answer;
503     char temp [MAX];
504 
505     printf ("\n[Double Chance]\n\n");
506 
507     while (stop == 0 && i < 2)
508     {
509         if (i == 0)
510         {
511             printf ("> Answer: ");
512         }
513         else
```

```
514         {
515             printf ("\n> Try again: ");
516         }
517         fgets (temp, MAX, stdin);
518         answer = atoi (temp);
519 
520         if (answer == correct)
521         {
522             stop = 1;
523         }
524         i ++;
525     }
526     if (stop == 1)
527     {
528         correct = 1;
529     }
530     else
531     {
532         correct = 2;
533     }
534     return correct;
535 }
536 int Skip (int cat, int play, abil_t *abil)
537 {
538     int correct, answer;
539     char temp [MAX];
540 
541     quest_p quest;
542 
543     printf ("\n[Skip]\n\n");
544 
545     quest = Quests_Read (cat);
546     printf ("%s\n", quest -> question);
547 
548     correct = Options (1, play, abil, quest);
549 
550     printf ("\n> Answer: ");
551     fgets (temp, MAX, stdin);
552     answer = atoi (temp);
553 
554     if (answer == correct)
555     {
556         correct = 1;
557     }
558     else
559     {
560         correct = 2;
561     }
562     return correct;
563 }
564 int Summon_Ally (int correct, abil_t *abil)
565 {
566     int opinion, answer;
567     char temp [MAX];
568 
569     if (rand () %10 < 7)
570     {
571         opinion = correct;
```

```
572     }
573     else
574     {
575         opinion = (rand () %3) + 1;
576 
577         while (opinion == correct)
578         {
579             opinion = (rand () %3) + 1;
580         }
581     }
582     printf ("\n[Ally]\n\n\"I think the correct answer is the option %d.!\"\n\n",
            opinion);
583 
584     printf("> Answer: ");
585     fgets (temp, MAX, stdin);
586     answer = atoi(temp);
587 
588     if (answer == correct)
589     {
590         correct = 1;
591     }
592     else
593     {
594         correct = 2;
595     }
596     return correct;
597 }
598 int Elite (int lv, int error, stat_t *stat, elite_t *elite)
599 {
600     int i, used_elite = 0, avail_elite = 0, n;
601     char temp [MAX];
602 
603     printf ("[Opponents]\n");
604 
605     for (i = 0; i < 6; i ++)
606     {
607         if (elite[i].use == 1)
608         {
609             used_elite ++;
610         }
611         else
612         {
613             avail_elite ++;
614         }
615     }
616     if (used_elite > 0)
617     {
618         printf ("\n[Defeated]\n\n");
619 
620         for (i = 0; i < 6; i ++)
621         {
622             if (elite[i].use == 1)
623             {
624                 printf ("* %s (%d pts)\n", elite[i].cat, stat -> correct [i]);
625             }
626         }
627     }
628     if (avail_elite > 0)
```

```
629     {
630         printf ("\n[Available]\n\n");
631 
632         for (i = 0; i < 6; i ++)
633         {
634             if (elite[i].use == 0)
635             {
636                 printf ("%d. %s (%s) (%d pts)\n", i + 1 , elite[i].name[lv], elite[i
                        ].cat, stat -> correct [i]);
637             }
638         }
639     }
640     if (error == 0)
641     {
642         printf ("\n> Number: ");
643     }
644     else
645     {
646         printf ("\n> Please, choose a valid number: ");
647     }
648     fgets (temp, sizeof (temp), stdin);
649     n = atoi (temp);
650 
651     system ("clear");
652     return (n - 1);
653 }
654 int Questions (int choose, int play, abil_t *abil)
655 {
656     int correct, answer = 0;
657     char temp [MAX];
658 
659     quest_p quest = NULL;
660 
661     quest = Quests_Read (choose);
662 
663     printf("(!) %s\n", quest -> question);
664 
665     correct = Options (0, play, abil, quest);
666 
667     printf ("\n> Answer: ");
668     fgets (temp, MAX, stdin);
669     answer = atoi (temp);
670 
671     switch (answer)
672     {
673         case 0: correct = 0;
674                 break;
675 
676         case 5: if (abil[0].use > 0)
677                 {
678                     abil[0].use --;
679                     correct = Bomb (correct, abil);
680                 }
681                 else
682                 {
683                   correct = 2;
684                 }
685                 break;
```

```
686
687         case 6: if (abil[1].use > 0)
688                 {
689                     abil[1].use --;
690                     correct = Double_Chance (correct, abil);
691                 }
692                 else
693                 {
694                   correct = 2;
695                 }
696                 break;
697
698         case 7: if (abil[2].use > 0)
699                 {
700                     abil[2].use --;
701                     correct = Skip (choose, play, abil);
702                 }
703                 else
704                 {
705                   correct = 2;
706                 }
707                 break;
708
709         case 8: if (abil[3].use > 0)
710                 {
711                     abil[3].use --;
712                     correct = Summon_Ally (correct, abil);
713                 }
714                 else
715                 {
716                   correct = 2;
717                 }
718                 break;
719
720         default: if (answer == correct)
721                 {
722                     correct = 1;
723                 }
724                 else
725                 {
726                     correct = 2;
727                 }
728                 break;
729     }
730     return correct;
731 }
732 void Status (int hp_p, int hp_e, int hp_e_f, int lv, int choose, int play, int atk_p
        , int streak, int atk_e, char name_e [], char cat [], stat_t *stat, pwrup_t *
        pwrup)
733 {
734     int i, j;
735
736     system ("clear");
737
738     if (hp_p > 100)
739     {
740         hp_p == 100;
741     }
```

```
742     if (hp_e > hp_e_f)
743     {
744         hp_e == hp_e_f;
745     }
746     printf ("\n< %s >", stat -> name);
747 
748     for (i = 0; i != 21 - strlen (stat -> name); i ++)
749     {
750         printf (" ");
751     }
752     if (play == 1)
753     {
754         printf ("\t[Elite Six]\t");
755     }
756     else if (play == 2)
757     {
758         printf ("\t[Survival]\t");
759     }
760     else
761     {
762         printf ("\t[Hardcore]\t");
763     }
764     if (choose != -1)
765     {
766         printf ("< %s", name_e);
767 
768         if (choose != -2)
769         {
770             printf (" | %s (%d pts)", cat, stat -> correct [choose]);
771         }
772         printf (" >");
773     }
774     printf ("\n[HP: ");
775 
776     for (j = 0; j <= 10; j ++)
777     {
778         if (hp_p >= j * 10 && hp_p < (j + 1) * 10)
779         {
780             if (hp_p >= 1 && hp_p <= 10)
781             {
782                 i = 0;
783                 j = 1;
784             }
785             for (i = 0; i < j; i ++)
786             {
787                 printf ("#");
788             }
789             for (i = j; i < 10; i ++)
790             {
791                 printf ("-");
792             }
793         }
794     }
795     printf ("] (%d/100)", hp_p);
796 
797     if (play != 2)
798     {
799         printf ("\t[Level %d]\t", lv);
```

```
800     }
801     else
802     {
803         printf ("\t[Round %d]\t", lv);
804     }
805     if (choose != -1)
806     {
807         printf ("[HP: ");
808 
809         for (j = 0; j <= 10; j ++)
810         {
811             if (hp_e >= j * hp_e_f / 10 && hp_e < (j + 1) * hp_e_f / 10)
812             {
813                 if (hp_e >= 1 && hp_e <= hp_e_f / 10)
814                 {
815                     i = 0;
816                     j = 1;
817                 }
818                 for (i = 0; i < j; i ++)
819                 {
820                     printf ("#");
821                 }
822                 for (i = j; i < 10; i ++)
823                 {
824                     printf ("-");
825                 }
826             }
827         }
828         printf ("] (%d/%d)", hp_e, hp_e_f);
829     }
830     printf ("\n(ATK: %d)", atk_p);
831 
832     printf ("\t\t\t[Streak: %d]\t", streak);
833 
834     if (choose != -1)
835     {
836         printf ("(ATK: %d)", atk_e);
837     }
838     printf ("\n");
839 
840     for (i = 1; i < 7; i ++)
841     {
842         if (pwrup[i].use == 1)
843         {
844             printf ("(%s) ", pwrup[i].abrev);
845         }
846     }
847     printf("\n\n");
848 }
849 int Play (int play, stat_t *stat, pwrup_t *pwrup, abil_t *abil, elite_t *elite,
        minion_t *minion)
850 {
851     int i, finish = 0, lv = 1, streak = 0, choose = -1, error, acc_p, acc_e, crit,
            correct;
852     float atk_p, hp_p = 100, hp_e, hp_e_f, atk_e, dmg_e, dmg_p;
853     char name_e [MAX], cat [MAX];
854 
855     for (i = 0; i < 4; i ++)
```

```
856     {
857         if (play == 1)
858         {
859             abil[i].use = 1;
860         }
861         else
862         {
863             abil[i].use = 0;
864         }
865     }
866     for (i = 0; i < 7; i ++)
867     {
868         pwrup[i].use = 0;
869     }
870     for (i = 0; i < 6; i ++)
871     {
872         elite[i].use = 0;
873     }
874     while (finish == 0)
875     {
876         atk_p = 20 + 10 * (pwrup[5].use);
877         Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e, name_e,
                cat, stat, pwrup);
878
879         if (play != 2 && lv < 7)
880         {
881             choose = Elite (lv - 1, 0, stat, elite);
882             while (choose < 0 || choose > 5 || elite[choose].use == 1)
883             {
884                 choose = -1;
885                 Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                        name_e, cat, stat, pwrup);
886                 choose = Elite (lv - 1, 1, stat, elite);
887             }
888             elite[choose].use = 1;
889             strcpy (name_e, elite[choose].name [lv - 1]);
890             strcpy (cat, elite[choose].cat);
891         }
892         hp_e_f = 75 + 25 * lv;
893         hp_e = hp_e_f;
894
895         atk_e = 10 + 5 * lv;
896
897         if (play == 2)
898         {
899             choose = rand () %6;
900             hp_e_f = 10;
901             hp_e = 10;
902             strcpy (name_e, minion[choose].name [rand () %10]);
903             strcpy (cat, minion[choose].cat);
904         }
905         if (play != 2 && lv == 7)
906         {
907             hp_e_f = 300;
908             hp_e = 300;
909             atk_e = 50;
910             strcpy (name_e, "Champion EdKeriohn");
911             choose = -2;
```

```
912         }
913         Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e, name_e,
                cat, stat, pwrup);
914
915         if (play == 1)
916         {
917             error = Powerups (0, pwrup, abil);
918             while (error == 1)
919             {
920                 Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                        name_e, cat, stat, pwrup);
921                 error = Powerups (1, pwrup, abil);
922             }
923         }
924         atk_p = 20 + 10 * (pwrup[5].use);
925         if (play == 2)
926         {
927             pwrup[1].use = 1;
928             pwrup[3].use = 1;
929             pwrup[4].use = 1;
930             pwrup[6].use = 1;
931         }
932         while (hp_p >= 1 && hp_e >= 1)
933         {
934             if (pwrup[1].use == 0)
935             {
936                 acc_p = rand () %5;
937             }
938             else
939             {
940                 acc_p = 1;
941             }
942             acc_e = rand () %5;
943
944             if (rand () %10 == 2)
945             {
946                 crit = 2;
947             }
948             else
949             {
950                 crit = 1;
951             }
952             dmg_e = atk_p * crit * ((rand () %16) + 85) / 100;
953
954             if (pwrup[4].use == 0)
955             {
956                 dmg_p = atk_e * crit * ((rand () %16) + 85) / 100;
957             }
958             else
959             {
960                 dmg_p = 2 * atk_e / 3 * crit * ((rand () %16) + 85) / 100;
961             }
962             if (play != 2 && lv == 7)
963             {
964                 choose = rand () %6;
965                 strcpy (cat, elite[choose].cat);
966             }
```

```
967             Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                    name_e, cat, stat, pwrup);
968
969             correct = Questions (choose, play, abil);
970             if (correct == 1)
971             {
972                 sleep (2);
973                 printf ("\n(!) Correct!\n\n");
974                 streak ++;
975                 if (streak > stat -> streak)
976                 {
977                     stat -> streak = streak;
978                 }
979                 stat -> correct [choose] ++;
980                 sleep (2);
981                 Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                        name_e, cat, stat, pwrup);
982                 sleep (2);
983                 if (acc_p != 0)
984                 {
985                     hp_e -= dmg_e;
986                     if (hp_e < 0)
987                     {
988                         hp_e = 0;
989                     }
990                     hp_p += 0.5 * dmg_e * pwrup[6].use;
991                     if (hp_p > 100)
992                     {
993                         hp_p = 100;
994                     }
995                     if (crit == 2)
996                     {
997                         printf ("(!) Critical hit!\n\n");
998                         sleep (2);
999                     }
1000                    Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak,
                            atk_e, name_e, cat, stat, pwrup);
1001                    if (crit == 2)
1002                    {
1003                        printf ("(!) Critical hit!\n\n");
1004                    }
1005                    printf ("(!) You took %.1f HP!\n\n", dmg_e);
1006                    sleep (3);
1007                }
1008                else
1009                {
1010                    printf ("(!) Attack missed...\n\n");
1011                    sleep (3);
1012                }
1013            }
1014            else if (correct == 2)
1015            {
1016                sleep (2);
1017                printf ("\n(!) Wrong!\n\n");
1018                streak = 0;
1019                sleep (2);
1020                Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                        name_e, cat, stat, pwrup);
```

```
1021                sleep (2);
1022                if (acc_e != 0)
1023                {
1024                    hp_p -= dmg_p;
1025                    if (hp_p < 0)
1026                    {
1027                        hp_p = 0;
1028                    }
1029                    if (crit == 2)
1030                    {
1031                        printf ("(!) Critical hit!\n\n");
1032                        sleep (2);
1033                    }
1034                    Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak,
                        atk_e, name_e, cat, stat, pwrup);
1035                    if (crit == 2)
1036                    {
1037                        printf ("(!) Critical hit!\n\n");
1038                    }
1039                    printf ("(!) You lost %.1f HP!\n\n", dmg_p);
1040                    sleep (3);
1041                }
1042                else
1043                {
1044                    printf ("(!) Attack missed...\n\n");
1045                    sleep (3);
1046                }
1047            }
1048            else
1049            {
1050                hp_p = 0;
1051            }
1052            if (hp_e == 0)
1053            {
1054                Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                    name_e, cat, stat, pwrup);
1055                printf ("(!) You have defeated %s!\n\n", name_e);
1056                if (play != 2)
1057                {
1058                    if (lv == 7)
1059                    {
1060                        stat -> defeated [6] ++;
1061                    }
1062                    else
1063                    {
1064                        stat -> defeated [lv - 1] ++;
1065                    }
1066                }
1067                else
1068                {
1069                    stat -> defeated [7] ++;
1070                }
1071                sleep (3);
1072
1073                if (play != 2 && lv == 7)
1074                {
1075                    printf ("(!) Congratulations! You are now the Champion of the
                        Elite Six!\n\n");
```

```
1076                    sleep (3);
1077                    finish = 1;
1078                }
1079            }
1080            if (hp_p < 1)
1081            {
1082                Status (hp_p, hp_e, hp_e_f, lv, choose, play, atk_p, streak, atk_e,
                        name_e, cat, stat, pwrup);
1083                printf ("(!) You have been defeated by %s...\n\n", name_e);
1084                sleep (3);
1085                printf ("(!) Game over!\n\n");
1086                sleep (3);
1087                hp_p = 0;
1088                finish = 1;
1089            }
1090            else
1091            {
1092                hp_p += 5 * pwrup[3].use;
1093                if (hp_p > 100)
1094                {
1095                    hp_p = 100;
1096                }
1097            }
1098        }
1099        if (hp_p > 0)
1100        {
1101            hp_p += 50 * pwrup[2].use;
1102        }
1103        if (hp_p > 100)
1104        {
1105            hp_p = 100;
1106        }
1107        if (lv - 1 > stat -> lv [play - 1])
1108        {
1109            stat -> lv [play - 1] = lv - 1;
1110        }
1111        lv ++;
1112    }
1113 }
1114 int New_Game (stat_t *stat, pwrup_t *pwrup, abil_t *abil, elite_t *elite, minion_t *
        minion)
1115 {
1116     int n = -1, error = 0;
1117     char temp [MAX];
1118
1119     while (n != 0)
1120     {
1121         system ("clear");
1122
1123         printf ("\n[New game]\n\n");
1124
1125         printf ("[Game modes]\n\n");
1126
1127         printf ("1. Elite Six:\n");
1128         printf ("   Defeat each member of the Elite Six, each one specialized in a
                category.\n");
1129         printf ("   If you defeat all of them, you may challenge the Champion of the
                 Elite specialized in every category.\n\n");
```

```
1130            printf ("2. Survival:\n");
1131            printf ("   Face an infinite wave of Minions and stay alive for the longest
                    number of rounds.\n");
1132            printf ("   You have some unlocked power-ups with you but you can't use
                    abilities.\n\n");
1133            printf ("3. Hardcore:\n");
1134            printf ("   Elite Six game mode with no power-ups nor abilities.\n\n");
1135
1136            printf ("0. Cancel.\n\n");
1137
1138            if (error == 0)
1139            {
1140                printf ("> Number: ");
1141            }
1142            else
1143            {
1144                printf ("> Please, choose a valid number: ");
1145            }
1146            fgets (temp, sizeof(temp), stdin);
1147            n = atoi (temp);
1148
1149            if (n >= 1 && n <= 3)
1150            {
1151                Play (n, stat, pwrup, abil, elite, minion);
1152            }
1153            else
1154            {
1155                error = 1;
1156            }
1157        }
1158        system ("clear");
1159    }
1160    void Name (stat_t *stat)
1161    {
1162        char name_temp [MAX];
1163
1164        system ("clear");
1165
1166        printf ("\n[Changing name]\n\n");
1167        printf ("> Name: ");
1168        fgets (name_temp, sizeof (name_temp), stdin);
1169        name_temp [strlen (name_temp) - 1] = '\0';
1170
1171        while (strlen (name_temp) > 21)
1172        {
1173            system ("clear");
1174            printf ("\n[Changing name]\n\n");
1175            printf ("> Your name must have less than 21 characters: ");
1176            fgets (name_temp, sizeof (name_temp), stdin);
1177            name_temp [strlen (name_temp) - 1] = '\0';
1178        }
1179        strcpy (stat -> name, name_temp);
1180
1181        system ("clear");
1182    }
1183    void Delete (stat_t *stat)
1184    {
1185        int i;
```

```
1186
1187      for (i = 0; i < 6; i ++)
1188      {
1189          stat -> correct [i] = 0;
1190      }
1191      for (i = 0; i < 8; i ++)
1192      {
1193          stat -> defeated [i] = 0;
1194      }
1195      for (i = 0; i < 3; i ++)
1196      {
1197          stat -> lv [i] = 0;
1198      }
1199      stat -> streak = 0;
1200
1201      system ("clear");
1202  }
1203  void Statistics (int error, stat_t *stat, elite_t *elite)
1204  {
1205      int i, correct = 0, defeated = 0, n;
1206      char temp [MAX];
1207
1208      system ("clear");
1209
1210      printf ("\n[Statistics]\n\n");
1211
1212      printf ("[Name]: %s\n\n", stat -> name);
1213
1214      for (i = 0; i < 6; i ++)
1215      {
1216          correct += stat -> correct [i];
1217      }
1218      printf ("[Correct answers]: %d\n\n", correct);
1219
1220      for (i = 0; i < 6; i ++)
1221      {
1222          printf ("* %s: %d\n", elite[i].cat, stat -> correct [i]);
1223      }
1224      for (i = 0; i < 8; i ++)
1225      {
1226          defeated += stat -> defeated [i];
1227      }
1228      printf ("\n[Defeated opponents]: %d\n\n", defeated);
1229
1230      printf ("* Elite (1-6): %d", stat -> defeated [0]);
1231      for (i = 1; i < 6; i ++)
1232      {
1233          printf (", %d", stat -> defeated [i]);
1234      }
1235      printf ("\n* Champion: %d\n", stat -> defeated [i]);
1236      printf ("* Minions: %d\n\n", stat -> defeated [i + 1]);
1237
1238      printf ("[Game modes (records)]\n\n");
1239
1240      printf ("* Elite Six: level %d", stat -> lv [0]);
1241      if (stat -> lv [0] == 7)
1242      {
1243          printf (" (completed)");
```

```
1244      }
1245      printf ("\n* Survival: %d rounds\n", stat -> lv [1]);
1246      printf ("* Hardcore: level %d", stat -> lv [2]);
1247      if (stat -> lv [2] == 7)
1248      {
1249          printf (" (completed)");
1250      }
1251      printf ("\n\n[Biggest correct answer-streak]: %d\n\n", stat -> streak);
1252
1253      printf ("1. Change name\n");
1254      printf ("2. Delete progress\n\n");
1255      printf ("0. Cancel\n\n");
1256
1257      if (error == 0)
1258      {
1259          printf ("> Number: ");
1260      }
1261      else
1262      {
1263          printf ("> Please, choose a valid number: ");
1264      }
1265      fgets (temp, sizeof (temp), stdin);
1266      temp [strlen (temp) - 1] = '\0';
1267      n = atoi (temp);
1268
1269      switch (n)
1270      {
1271          case 0: system ("clear");
1272                  break;
1273
1274          case 1: Name (stat);
1275                  break;
1276
1277          case 2: Delete (stat);
1278                  break;
1279
1280          default: Statistics (1, stat, elite);
1281                   break;
1282      }
1283  }
1284  void Credits ()
1285  {
1286      system ("clear");
1287      printf ("\n[Credits]\n\n");
1288      sleep (1);
1289      printf ("< The QUIZZER by EdKeriohn v.1.0 >\n\n");
1290      sleep (1);
1291      printf ("[Developer team]\n\n");
1292      printf ("* Leandro Alves - A82157\n");
1293      printf ("* J. Eduardo Santos - A82350\n\n");
1294      sleep (3);
1295      printf ("[Former member]\n\n");
1296      printf ("* David Machado - A82381\n\n");
1297      sleep (2);
1298      printf ("[Software used]\n\n");
1299      printf ("* Sublime Text - Code writting (C)\n");
1300      printf ("* Texmaker - Report writting (LaTeX)\n");
1301      printf ("* ubuntu 16.04 LTS - Operative system\n\n");
```

```
1302      sleep (4);
1303      printf ("[Based on]\n\n");
1304      printf ("* Pokemon - Combat mechanics and Elite Six game mode\n");
1305      printf ("* Team Fortress 2 - Power-ups\n");
1306      printf ("* Trivia Crack - Categories, abilities and questions\n\n");
1307      sleep (4);
1308      printf ("[Development time]\n\n");
1309      printf ("* 2 weeks\n");
1310      printf ("* Release date: June 11th, 2017\n\n");
1311      sleep (3);
1312      printf ("[Other]\n\n");
1313      printf ("* Metodos de Programacao II\n");
1314      printf ("* Mestrado Integrado em Engenharia de Telecomunicacoes e Informatica\n"
              );
1315      printf ("* University of Minho\n\n");
1316      sleep (4);
1317      printf ("< EdKeriohn 2017 >\n\n");
1318      sleep (3);
1319      system ("clear");
1320  }
1321  int main ()
1322  {
1323      int n = -1, error = 0;
1324      char temp [MAX];
1325
1326      stat_t stat;
1327      pwrup_t pwrup [MAX] = {0};
1328      abil_t abil [MAX] = {0};
1329      elite_t elite [MAX] = {0};
1330      minion_t minion [MAX] = {0};
1331
1332      Stats_Read (&stat);
1333      Pwrups_Read (pwrup);
1334      Abils_Read (abil);
1335      Elite_Read (elite);
1336      Minions_Read (minion);
1337
1338      srand (time (NULL));
1339
1340      system ("clear");
1341
1342      printf ("\n  (!) Hi and welcome to The QUIZZER by EdKeriohn.\n");
1343      printf ("(!) After one month in development, it is worth the wait.\n");
1344      printf ("\t    (!) Thanks and have fun!\n");
1345
1346      while (n != 0)
1347      {
1348          printf ("\n");
1349          printf ("****** #### #  # ### ##### ##### #### #### ******\n");
1350          printf (" ***** #  # #  #  #  #   # #   # #    #  # *****\n");
1351          printf ("  **** #  # #  #  #     ##    ## #### #  # ****\n");
1352          printf ("   *** #  # #  #  #  ###   ###   #    ###  ***\n");
1353          printf ("    ** ### #  #  #  #   # #   # #    #  # **\n");
1354          printf ("     *   ##  ##  ### ##### ##### #### #  # *\n\n");
1355
1356          printf ("\t\t [Main Menu]\n\n");
1357
1358          printf ("\t\t 1. New game\n");
```

```
1359            printf ("\t\t2. Statistics\n");
1360            printf ("\t\t 3. Credits\n\n");
1361            printf ("\t\t  0. Exit\n\n");
1362
1363            if (error == 0)
1364            {
1365                printf ("\t\t > Number: ");
1366            }
1367            else
1368            {
1369                printf ("\t> Please, choose a valid number: ");
1370            }
1371            fgets (temp, sizeof (temp), stdin);
1372            n = atoi (temp);
1373
1374            switch (n)
1375            {
1376                case 1: New_Game (&stat, pwrup, abil, elite, minion);
1377                    error = 0;
1378                        break;
1379
1380                case 2: Statistics (0, &stat, elite);
1381                    error = 0;
1382                        break;
1383
1384                case 3: Credits ();
1385                    error = 0;
1386                        break;
1387
1388                default: error = 1;
1389                         system ("clear");
1390                         break;
1391            }
1392        }
1393        system ("clear");
1394        printf ("\n(!) Thank you for playing The QUIZZER by EdKeriohn!\n");
1395        printf ("\n(!) Please, come back again! ;)\n\n");
1396        sleep (4);
1397        system ("clear");
1398        Stats_Write (&stat);
1399
1400        return 0;
1401    }
```

6.2 Ficheiros adicionais

Os ficheiros necessários para o funcionamento adequado do programa foram enviados numa pasta *.zip* e devem estar na mesma pasta do ficheiro *C*.

7 Conclusão

Em suma, as tarefas foram cumpridas e conjugadas com êxito, sendo que o resultado final se revela um quanto fruitivo.

Foi, no geral, um trabalho extremamente enriquecedor e altamente pedagógico uma vez que conjugou o aprefeiçoamento em *C* (principalmente no manuseamento em estruturas, ficheiros sequênciais, listas ligadas, apontadores, memória alocada dinamicamente e listas generalizadas), em *L*inux e em LaTeX.