Trabalho Prático II - TEP - PokeTep

Prof. Vinicíus Mota DI/UFES

Data de entrega: 12/05/2021 Locais de entrega: - submetido pelo Repl.it; - E também pelo AVA.

Consideração inicial

Descrições de trabalhos podem conter erros (acho que experienciamos isso de primeira mão esse semestre). É muito importante que esses erros sejam corrigidos o mais cedo possível. Leia essa descrição com calma e, caso tenha dúvidas ou algum erro para reportar, fale com os monitores o mais rápido possível!

1. Objetivos

Este segundo trabalho visa o desenvolvimento das habilidades de manipulação de callbacks e estruturas encadeadas simples, bem como criação de TADs, bibliotecas, organização e documentação de código. O objetivo, agora, é criar uma versão do famoso jogo da Game Freak Pokémon! Nessa versão, o jogo roda completamente no terminal e tem suas funcionalidades simplificadas em relação às publicações originais.

Visão Geral do jogo

Original:

O jogo original é um RPG e sua mecânica principal consiste em um mapa por onde o jogador anda e, aleatoriamente, tem um encontro com um inimigo (pokemon). Neste momento, é iniciada uma batalha entre o jogador e o pokemon inimigo, que consiste em uma mecânica de turnos entre o jogador e o oponente. O jogador não luta diretamente contra o pokemon, mas usa os seus próprios pokemons para lutar contra o oponente. A batalha acaba quando o jogador derrotar o pokemon inimigo, capturar o pokemon inimigo ou fugir da batalha. A batalha também acaba se o pokemon inimigo derrotar todos os pokemons do jogador (o jogador pode ter varios pokemon que podem ser selecionados em qualquer momento da batalha).

O jogo original possui outras mecânicas, uma história e objetivos maiores, mas para fins desse tabalho, essas são as informações relevantes. Caso queira saber mais sobre a série de jogos, confira esse artigo da wikipedia.

Simplificação para este TP:

No nosso trabalho, não teremos esse mapa. Na verdade, o jogador não poderá se mover, mas o seu caminho já estará definido. Na versão deste TP, o jogo consistirá apenas das batalhas entre os pokemons do jogador e o pokemon inimigo. O jogador vai começar com 3 pokemons e 3 pokebolas e, logo em seguida, começa um caminho de tamanho indefinido em que cada parada terá um pokemon aleatório esperando. Nesse momento, a batalha vai começar, onde o jogador terá algumas opções do que fazer. O jogo vai acabar quando todos os pokemons do jogador forem derrotados.

IMPORTANTE:

- O código deve ser devidamente separado em bibliotecas com os devidos cabeçalhos e implementações, sempre levando em consideração as boas práticas.
- É responsabilidade do desenvolvedor:
  - Garantir que o programa faz uso eficiente da memória.
  - Filtrar erros de entrada dos usuários. Mesmo que não estejam listados aqui.
  - Sempre comentar essas verificações.
- A compilação do executável deve ser por meio de um Makefile.
- Código deve estar bem comentado.
- É opcional "limpar o terminal" entre as passagens do menu de opções.

2. Descrição do sistema

O nosso jogo consistirá em um caminho (de tamanho indefinido) que teremos que percorrer. Em cada parada desse caminho, um pokemon (aleatório) nos aguarda e, quando nos encontramos com ele, teremos um ambiente de batalha parecido com o do jogo original.

2.1 Inicialização

Assim como no TP I, aqui também teremos menus. Eles não precisam ser visualmente idênticos ao mostrado aqui, mas suas funcionalidades precisam ser, estritamente, as mesmas do que for indicado! Além disso, quando uma entrada for inválida, o programa deve pedir para o jogador informar uma opção válida.

O menu inicial é a primeira coisa que vemos ao executar o programa, e deve ter o seguinte formato:

```
1- Jogar
2- Melhores pontuações
3- Sair
```

Nesse momento, 1 deve nos levar à escolha de nome, 2 deve levar às melhores pontuações e 3 deve sair do programa (independente de como a informação está disposta na tela!). As pontuações serão apenas a quantidade de batalhas que os jogadores conseguiram vencer, ordenadas em ordem decrescente:

```
1- Melhor_Jogador: 100
2- Nome_bom: 98
3- Nao_sei_jogar: 5
```

Essas informações não precisam ser paginadas, mas elas devem estar presentes um um arquivo externo (mais sobre isso na seção sobre arquivos). O programa espera a leitura de uma quebra de linha (que pode ser obtida com a função `getc()`). Daí, o jogo volta para o menu incial.

Ao escolher a opção 1, deve aparecer a seguinte tela:

```
Qual o seu nome?
jogador10
```

Aqui o jogador deve digitar um nome com, apenas, caracteres alfabéticos ou com underlines (`_`) e sem espaços. Caso o nome seja inválido, pedir para digitar novamente:

```
Digite um nome válido!
jogador_dez
```

Após escrever um nome válido, o jogador deve então selecionar 3 pokemons de uma lista de 6 totais (que deverão ser implementados pela dupla) para começar.

```
1- Pikachu
2- Charizard
3- Venusaur
4- Blastoise
5- Steelix
6- Mew
```

Quando o jogador selecionar um dos pokemons, o menu vai ser redesenhado, agora com apenas cinco opções. Suponha que foi escolhido Pikachu. O novo menu deve ser:

```
1- Charizard
2- Venusaur
3- Blastoise
4- Steelix
5- Mew
```

A próxima escolha nos leva direto para a primeira batalha.

2.2 A batalha

A batalha segue um formato parecido com a do jogo original. Ela acontece entre o primeiro pokemon disponível do jogador (respeitando a ordem de seleção) e o pokemon aleatório da parada.

Quando um pokemon do usuário é derrotado, ele é removido da lista de pokemons disponíveis e o próximo dessa lista se torna o novo pokemon ativo. Ao longo do caminho, o jogador também pode tentar capturar os pokemons oponentes e, caso ele consiga, o novo pokemon vai para o final da lista.

Não existe um limite de quantos pokemons o jogador pode capturar, mas ele só consegue realizar a ação caso ele tenha pokebolas, das quais ele sempre começa com três e tem uma chance de ganhar uma na transição entre cada parada.

A tela da baltalha deve ser no seguinte formato (supondo uma batalha entre Pikachu e Blastoise):

```
Um Blastoise selvagem aparece!

Blastoise: 100% HP
Pikachu: 100% HP

Escolha um movimento:
1- Choque do Trovão
2- Bater
3- Onda de choque
4- Pokebola (3 disponíveis)
5- Fugir
```

Opções 1, 2 e 3 - São ataques e os mesmos devem ser implementados (mais na seção de ataques);

Opção 4 - O jogador deve arremessar uma pokebola. Quando isso ocorre, existe uma chance do pokemon ser capturado, terminando a batalha e seguindo para a próxima. Essa opção deve ser seguida de quantas pokebolas o jogador tem e, caso a quantidade seja 0, deve pedir para o jogador selecionar outra opção.

Opção 5 - O jogador tenta fugir da batalha, o que tem uma chance de acontecer ou não e, caso não aconteça, a ação usa o movimento do jogador.

Após cada ação, o menu é atualizado para informar o que aconteceu. O programa, então, aguarda uma entrada para prosseguir com a batalha. Essa entrada é apenas uma quebra de linha. Suponha, então, que o jogador escolhe a opção 1 e o oponente perde 30% de HP. O menu deve, primeiro, atualizar e mostrar o que aconteceu:

```
Pikachu usou Choque do Trovão

Blastoise: 70% HP
Pikachu: 100% HP
```

Nesse momento o programa espera o usuário apertar a tecla Enter (que gera a quebra de linha), o que dá o controle para a IA, que seleciona um movimento aleatório (no nosso caso, Canhão d'Água, que tira 20% de HP de pikachu) e essa ação é indicada na tela, bem como a anterior:

```
Blastoise usou Canhão de Água

Blastoise: 70% HP
Pikachu: 80% HP
```

Vale ressaltar que a IA apenas escolhe um dos três movimentos do pokemon que está controlando, e todos os efeitos que esses movimentos geram acontecem de forma normal.

O programa espera, novamente, a tecla Enter para continuar a batalha, que voltaria para o controle do jogador.

Após a batalha acabar, a tela indica se o jogador ganhou ou perdeu:

```
jogador_10 venceu!
```

E espera novamente pela tecla enter do jogador. Em seguida, avançamos para a próxima parada, onde haverá um novo pokemon e, consequentemente, uma nova batalha. Esse ciclo continua até que todos os nossos pokemons tenham sido derrotados.

Entre cada parada, os pokemons do jogador recuperam 10 pontos de HP, não podendo ultrapassar o limite máximo de cada um. Isso não ocorre para os pokemons que foram derrotados.

Quando todos os pokemons do jogador forem derrotados, o jogo mostra a seguinte tela após indicar que o jogador perdeu:

```
Fim de jogo!
jogador_10 sobreviveu 18 rodadas e está em 3º lugar no placar!
```

O jogo, novamente, espera a quebra de linha do usuário antes de retornar ao menu inicial.

2.3 Pokemons e movimentos

Obviamente, não é possível implementar todos os pokemons em nosso jogo, mas o que podemos fazer é implementar alguns. Nesse trabalho, deverão ser implementados 6 pokemons: Pikachu, Charizard, Venusaur, Blastoise, Dragonite e Mew. A tabela a seguir tem as informações importantes para serem usadas no trabalho sobre os pokemons (os ataques durante a batalha devem aparecer na ordem em que aparecem na tabela):

Pokemon	HP	Ataque	Defesa	Tipo	Ataques
Pikachu	200	110	100	Elétrico	Choque do Trovão, Onda de Choque, Bater
Charizard	260	160	150	Fogo	Lança Chamas, Dormir, Bater
Blastoise	280	180	200	Água	Arma de Água, Proteger, Bater
Venusaur	300	160	160	Planta	Pó de sono, Bomba de Semente, Dois Gumes
Steelix	280	170	400	Metal	Rabo de Ferro, Dormir, Cavar
Mew	320	200	200	Psíquico	Metronomo, Bater, Auto-Destruir

Nem todos os ataques causam dano. A seguinte tabela fornece as informações importantes sobre os ataques:

Ataque	Causa dano	Poder	MT	Efeito
Choque do Trovão	Sim	40	Sim	Causa dano no oponente, com 10% de chance de paralizar por um turno
Onda de Choque	Não	-	Sim	Paralisa o oponente por um turno
Bater	Sim	40	Não	Causa dano no oponente
Lança Chamas	Sim	90	Sim	Causa dano no oponente e tem 10% de chance de queimar
Dormir	Não	-	Não	O pokemon dorme por dois turnos, mas ao final recupera 100% do HP
Arma de Água	Sim	40	Sim	Causa dano no oponente
Proteger	Não	-	Não	Protege te todos os ataques no próximo turno

Ataque	Causa dano	Poder	MT	Efeito
Pó de Sono	Não	-	Sim	Coloca o oponente para dormir entre 1 a 3 turnos
Bomba de Semente	Sim	80	Sim	Causa dano no oponente
Dois Gumes	Sim	120	Não	Causa dano no oponente, mas causa 1/3 desse dano no usuário
Rabo de Ferro	Sim	100	Sim	Causa dano no oponente
Cavar	Sim	80	Não	Se esconde na terra ficando imune no primeiro turno, causa dano no segundo
Metronomo	-	-	-	Seleciona um ataque aleatório (de todos disponíveis) e o realiza
Auto-Destruir	Sim	200	Não	Causa muito dano ao oponente, mas o pokemon usuário é derrotado

Os status têm os seguintes resultados: - **Dormir:** O pokemon dorme, não podendo realizar nenhum ataque, mas pode levar dano - **Queimar:** O pokemon leva 1/16 da vida máxima até o final da batalha (esse status não é aplicado à pokemons de tipo fogo) - **Paralizar:** Similar a dormir, não realiza nenhum movimento mas leva dano

O dano causado por um ataque é sempre calculado pela seguinte fórmula:

```
Dano = ((14*poder*A/D)/50 + 2) * modificador
```

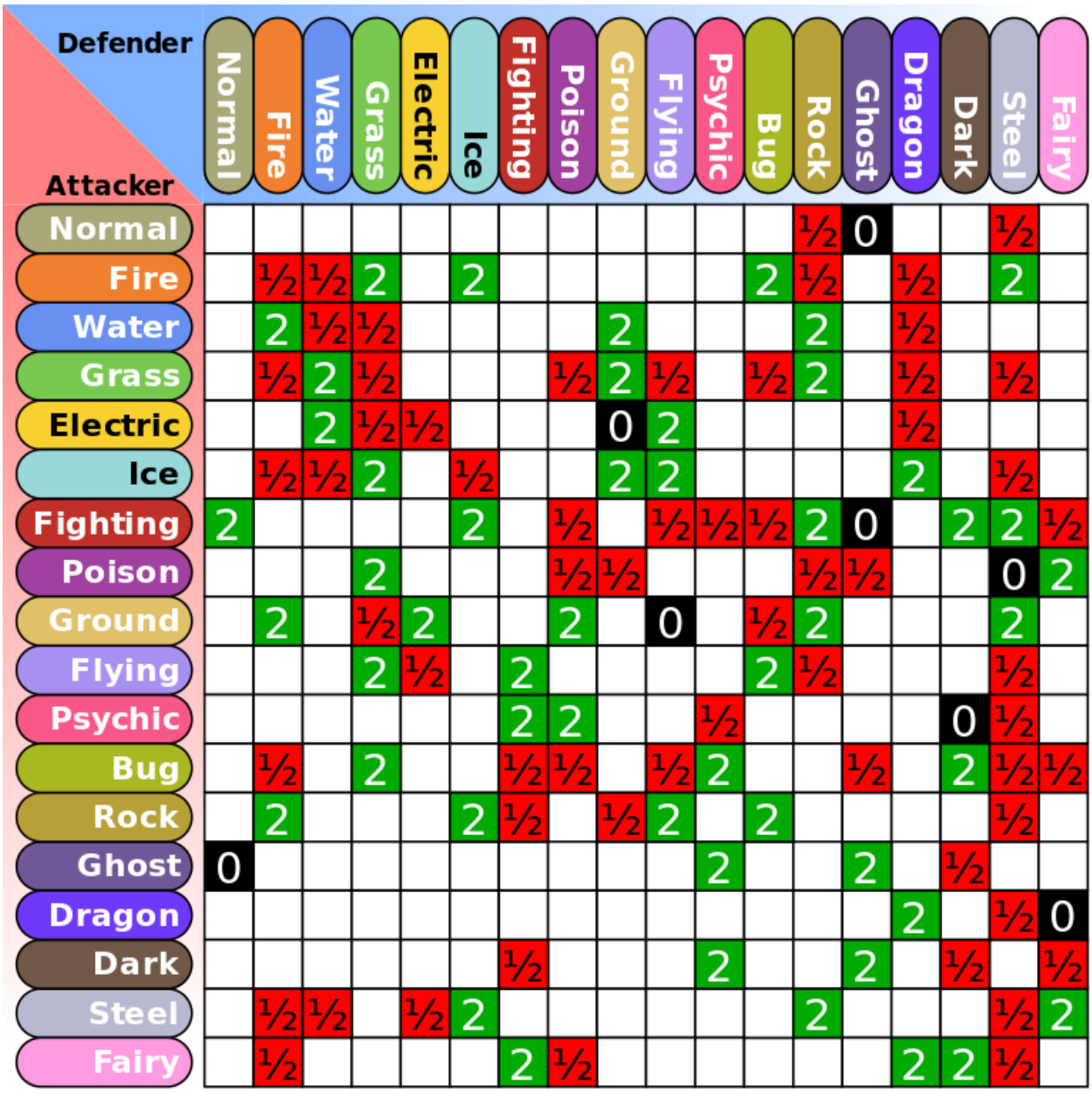
Onde: - `poder` é o poder do ataque; - `A` é o ataque do pokemon usando o ataque; - `D` é a defesa do pokemon alvo; - `modificador` é um valor calculado.

`modificador` é calculado por:

```
modificador = critico*MT*tipo
```

Onde: - `crítico` é um valor que tem uma chance de 1/24 de ser 2 e, caso contrário, é 1; - `MT` é 1.5 se o ataque tiver o mesmo tipo que o pokemon usuário e 1 caso o contrário; - `tipo` é a relação de tipo entre o pokemon usuário e o pokemon alvo.

Relações de tipo: As relações de tipo entre os pokemons são dadas pela seguinte tabela (células brancas representam 1):



O valor da relação depende de quem ataca e quem é alvo, então é preciso atenção ao selecionar o valor aplicado na fórmula!

2.4 Probabilidades

Alguns acontecimentos dependem de probabilidade para... Acontecer. A tabela seguinte lista os acontecimentos e como calcular suas probabilidades:

Acontecimento	Probabilidade
Fugir	C/4 (C = tentativas de fuga); Caso o pokemon ataque, 1/16
Conseguir uma Pokebola	C/12 (C = batalhas vencidas desde a última pokebola)
Ataque crítico	1/24
Pokemon aleatório é Mew	C/128 (C = batalhas vencidas desde o último Mew)
Lança chamas queima	1/10
Choque do trovão paralisa	1/10
Turnos do pó de sono	1/3 para cada possibilidade
Ataque metronomo	1/13 para cada possibilidade
Capturar um pokemon	(HPM/HPA)/20 (HPM = HP máximo do pokemon, HPA = HP atual)

Essas probabilidades devem ser calculadas para cada vez que ocorrer o acontecimento.

Vale lembrar que os valores de dano devem ser sempre números reais!

2.5 Arquivos de saída

O programa deve gerar alguns arquivos durante a execução. Esses arquivos **devem** estar exatamente com o mesmo formato que o especificado. Eles serão fornecidos pela linha de comando ao executar o programa.

2.5.1 Pontuações

Como mencionado em **2.1**, deve existir um arquivo contendo as melhores pontuações obtidas. Esse arquivo deve conter o nome e o número de batalhas ganhas pelo jogador na partida em questão:

```
<jogador1> <qtd_partidas>
<jogador2> <qtd_partidas>
<jogador3> <qtd_partidas>
.
.
.
```

A quantidade de jogadores não é definida. O arquivo deve ser ordenado, primeiro, pela pontuação em ordem decrescente e depois por ordem alfabética.

2.5.2 Log de batalhas

Para cada partida que ocorre, o programa deve anotar, em um arquivo, o que ocorre em cada turno de cada batalha. Esse log da batalha sempre começa com o número da partida, o número da batalha na partida e os dois pokemons que estão batalhando, no seguinte formato:

```
<partida.batalha>- <pokemon_jogador> vs <pokemon_maquina>
```

Em seguida, é listado cada moviento escolhido (começando com um `\t`) e, no caso de um ataque, o estado de cada pokemon (vida e seus status):

```
    <pokemon> usa <movimento>
    <pokemon1> <hp_pokemon1>% HP <(paralizado, dormindo, queimando)>
    <pokemon2> <hp_pokemon2>% HP <(paralizado, dormindo, queimando)>
```

Ou, caso contrário, a ação e se a ação fracassou ou não:

```
    Tentativa de <Captura ou Fuga>
    <Fracasso ou Sucesso>
```

Ao final de uma batalha, se o jogador venceu ou perdeu:

```
    <nome_do_jogador> <vence! ou perde!>
```

Se o jogador chegar no final da partida, deve ser impresso no arquivo:

```
Fim do jogo <numero_da_partida> <nome_do_jogador> sebreviveu <batalhas_vencidas> batalhas
```

Cada uma dessas informações deve ser impressa no arquivo de saída com uma quebra de linha extra as separando, bem como as diferentes partidas do mesmo arquivo. O log seguinte é válido para uma batalha:

```
1.1- Pikachu vs Blastoise
    Pikachu usa Choque do Trovão
    Pikachu: 100% HP
    Blastoise: 70% HP

    Blastoise usa Canhão de Água
    Pikachu: 80% HP
    Blastoise: 70% HP

    Pikachu usa Onda de Choque
    Pikachu: 80% HP
    Blastoise: 70% HP (paralizado)

    Tentativa Captura
    Fracasso

    Blastoise usa Super Chute
    Pikachu: 60% HP
    Blastoise: 70% HP

    Pikachu usa Choque do Trovão
    Pikachu: 60% HP
    Blastoise: 40% HP

    Blastoise usa Canhão de Água
    Pikachu: 40% HP
    Blastoise: 20% HP

    Tentativa Captura
    Sucesso

    jogador_dez vence!
```

3. Detalhes de implementação

Requisitos que devem estar presentes no seu código:

1. **Uso de TADs**

   Não é surpresa que, nesse trabalho, haverá forte uso de TADs novamente

2. **Uso de callbacks para a implementação dos diferentes ataques**

   O código do programa, idealmente, deve conter uma `struct` para o ataque do pokemon e, dentro dessa struct, uma função `callback` que realiza as interações entre os dois pokemons.

3. **Uso de estruturas encadeadas**

   Uma característica desse trabalho é que existem muitas informações com números indefinidos de componentes (a lista de pontuações, os pokemons disponíveis). Para guardar esses tipos de informação na memória, priorizem o uso de estruturas encadeadas!

Além desses requisitos, mantenham em mente as seguintes boas práticas:

- Comente o código bem (o que é diferente de comentar muito!);
- Fiquem de olho no `valgrind`! Não só pelo vazamento de memória, mas pelos acessos indevidos também (eles causam o segmentation fault!!!);
- É **PROIBIDO** o uso de variáveis globais! Exceção para *Dispatch tables*. Um código bem estruturado não precisa utilizar esse tipo de variável (muito menos a primitiva **goto**, que também está proibida);
- Pense nos TADs com carinho! Antes de começar a implementação, reflita sobre como vai implementar, em quais estruturas vai utilizar e como atender os requisitos do trabalho.

O trabalho também faz muito uso de números aleatórios, então é importante saber utilizar essa funcionalidade em C para obter as probabilidades corretas. Por exemplo, caso queira calcular uma probabilidade de 1/10, deve-se fazer:

```
float aleatorio = (float)rand()/(float)(RAND_MAX)
if(aleatorio <= 0.1){
    \\ vai acontecer
}
else{
    \\ não acontece
}
```

Para mais informações sobre os números aleatórios em C, de uma olhada nesse artigo do geeks for geeks.

4. Compilação e execução

Novamente, o trabalho deve ser compilado com um `Makefile` que deve gerar um arquivo executável `tp2`. Ao executar o trabalho, três argumentos devem ser passados:

- O diretório do arquivo de pontuações
- O diretório do arquivo de log das batalhas
- Um número inteiro que ser utilizado como *seed* para o gerador de números aleatórios do jogo ()

Uma execução do trabalho deve ser do tipo:

```
$ make
$ ./tp2 dir/para/placares.txt dir/para/logs.txt 42
```

Atenção: é importante converter o valor passado via argumento para inteiro antes de usar como *seed*.

5. Regras Gerais

O trabalho deverá ser feito **em dupla** e pelos próprios alunos, isto é, os alunos deverão necessariamente conhecer e dominar todos os trechos de código criados.

Cada dupla deverá trabalhar independente das outras, não sendo permitido a cópia ou compartilhamento de código. O professor irá fazer verificação automática de plágio. Trabalhos identificados como iguais, em termos de programação, serão penalizados com a nota zero. Isso também inclui a dupla que forneceu o trabalho, sendo, portanto, de sua obrigação a proteção de seu trabalho contra cópias ilícitas.

6. Entrega do Trabalho

O trabalho deverá ser submetido pelo Repl.it e também pelo AVA até as 23:59 do dia xx/05/2021.

O arquivo enviado pelo AVA deve seguir o padrão: TEP-2021_MATRICULA1_MATRICULA2.zip, substituindo a matrícula pelos respectivos números de matrícula da dupla.
O arquivo zip deve conter: Todos os arquivos necessários para compilação (cabeçalhos e códigos fontes) `Makefile` que compile o programa corretamente em um ambiente Linux. `README.md` contendo nome da dupla e com considerações de projeto que julgue necessário, incluindo as decisões de projeto tomadas. Por exemplo, explicação breve das bibliotecas, etc.

O código fonte deverá estar bem comentado, isto é, todas os structs e funções devem ter descrições claras do que ela faz, o que representa cada parâmetro e o que ela retorna (se houver). Pode-se adicionar também comentários que ajudem a entender a lógica da questão.

7. Avaliação

O trabalho será corrigido em dois turnos: correção do professor e entrevista. A correção do professor (CP) gerará uma nota entre 0 e 10 e as entrevistas (E) serão realizadas posteriormente à data de entrega do trabalho e será atribuída uma nota entre 0 e 1. Alunos que fizeram o trabalho em dupla farão a entrevista juntos porém terão seus desempenhos na entrevista avaliados de forma individual. Pequenas e pontuais correções de código serão permitidas no momento da entrevista, o que poderá acarretar em uma CP maior.

A nota final (NF) do trabalho será dada pela seguinte fórmula: NF = E * CP

O trabalho será pontuado de acordo com sua funcionalidade e outros aspectos de implementação, conforme as listas abaixo.

Aspectos funcionais

1. Implementar os pokemos e suas relações de tipo corretamente(10%)
2. Implementar os ataques de forma correta(10%)
3. Implementar a lógica de jogo corretamente (turnos, status, etc)(10%)
4. Implementar o placar corretamente(5%)
5. Gerar os arquivos corretamente(15%)

   Placar (5%) Log de batalhas (10%)

Aspectos de desenvolvimento:

1. Uso correto de TADs para abstração dos elementos envolvidos no trabalho. Menos *main* mais TADs! (15%)
2. Uso de callbacks (10%)
3. Uso de estruturas encadeadas (10%)
4. Executar o programa com o valgrind, sair corretamente e não ter vazamento de memória. (15%)