Documentação Jogo dos Troninhos

AUTORES
Cecilia Nascimento
Emerson Gouveia
Fabio Brum
Luis Carvalho
Samuel Assis
Terça, 11 de Junho de 2019

Índice Hierárquico

Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

Índice dos Componentes

Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Classes

Referência da Classe Arma

#include “Arma.h”

Membros Públicos

Arma ()
Arma (int _id, std::string _name, int _b_attack, int _price)
int **get_id** ()
void **set_id** (int _id)
int get_attack ()
void **set_attack** (int _attack)
std::string **get_name** ()
void **set_name** (std::string _name)
int get_price ()
void **set_price** (int _price)
void display_weapon ()
void **equip** (**Arma** toequip)
void **unequip** ()

Descrição detalhada

A classe arma foi criada para adicionar a mecânica de itens no jogo. Ela possui 4 atributos:1 string para o atributo name, 3 inteiros para os atributos id, b_attack e price.

Construtores e Destrutores

Arma::Arma () - constrói o atributo name com a string “Sem Arma” e name, b_attack e price com inteiros de valor 0 por default.
Arma::Arma (int *_id*, std::string *_name*, int *_b_attack*, int *_price*) - constrói o atributo name com a string “Sem Arma” e name, b_attack e price com inteiros de valor 0 por default.

Funções membros

void Arma::display_weapon () – Imprime os atributos do objeto

void Arma::equip (Arma *toequip*) - Recebe um objeto Arma e substitui o valor de todos os atributos atuais pelos atributos de toequip;

int Arma::get_attack ()

int Arma::get_id ()

std::string Arma::get_name ()

int Arma::get_price ()

void Arma::set_attack (int *_attack*)

void Arma::set_id (int *_id*)

void Arma::set_name (std::string *_name*)

void Arma::set_price (int *_price*)

void Arma::unequip () - Substitui o valor de todos os atributos atuais para os valores default (construtor sem argumentos);

Referência da Classe Armadura

#include “Armadura.h”

Membros Públicos

Armadura ()
Armadura (int id, std::string _name, int _b_defense, int price)
int **get_id** ()
void **set_id** (int _id)
int get_defense ()
void **set_defense** (int _b_defense)
std::string **get_name** ()
void **set_name** (std::string _name)
int get_price ()
void **set_price** (int _price)
void display_armor ()
void **equip** (**Armadura** toequip)
void **unequip** ()

Descrição detalhada

A classe Armadura foi criada para adicionar a mecânica de itens no jogo. Ela possui 4 atributos: 1 string para o atributo name, 3 inteiros para os atributos id, b_defense e price.

Construtores e Destrutores

Armadura::Armadura (): Constrói o atributo name com a string “Sem Armadura” e name, b_defense e price com inteiros de valor 0 por default.

Armadura::Armadura (int *id*, std::string *_name*, int *_b_defense*, int *price*): Constrói o atributo name com a string _name, e id, b_defense, price com os parâmetros recebidos, _id, _b_defense e _price, respectivamente.
Observações: se algum dos parâmetros inteiros acima forem negativos, o construtor constrói o valor correspondente com um número inteiro de valor 0.

Funções membros

void Armadura::display_armor (): Imprime os atributos do objeto

void Armadura::equip (Armadura *toequip*): Recebe um objeto Armadura e substitui o valor de todos os atributos atuais pelos atributos de toequip;

int Armadura::get_defense ()

int Armadura::get_id ()

std::string Armadura::get_name ()

int Armadura::get_price ()

void Armadura::set_defense (int *_b_defense*)

void Armadura::set_id (int *_id*)

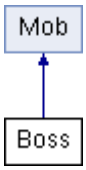
void Armadura::set_name (std::string *_name*)

void Armadura::set_price (int *_price*)

void Armadura::unequip () - Substitui o valor de todos os atributos atuais para os valores default (construtor sem argumentos);

Referência da Classe Boss

#include <boss.h>
Diagrama de hierarquia para Boss:



Membros Públicos

Boss (std::string, float, float, float, float)
virtual float **get_max_attack** () override
virtual float **get_min_attack** () override
virtual float **get_defense** () override
virtual std::string **get_name** () override
virtual float **get_life** () override
int get_hdamage ()
int get_hspend ()
virtual void **set_max_attack** (float) override
virtual void **set_min_attack** (float) override
virtual void **set_defense** (float) override
virtual void **set_life** (float) override
virtual int **type** () override
void set_skill (Habilidade)

Descrição detalhada

A classe é uma sub-classe de Mob usada na batalha final de cada fase é um mob mais forte e com habilidade que é usada na batalha, tem como atributos *life, max_attack, min_attack, defense* que são valores *float, name* que é uma *string. Possui também um* atributo *skill* que é da classe Habilidade.

Construtores e Destrutores

- **Boss::Boss (std::string , float , float , float , float) -** Construtor da classe que recebe uma *string* para nome e quatro valores *float* para os atributos de vida, defesa, ataque maximo e ataque minimo.

Funções membros

virtual float Boss::get_defense ()[override], [virtual]

Implementa **Mob** (*p.15*).

int Boss::get_hdamage ()

int Boss::get_hspend ()

virtual float Boss::get_life ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual float Boss::get_max_attack ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual float Boss::get_min_attack ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual std::string Boss::get_name ()[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Boss::set_defense (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Boss::set_life (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Boss::set_max_attack (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Boss::set_min_attack (float)[override], [virtual]

Implementa **Mob** (*p.16*).

void Boss::set_skill (Habilidade) - O método recebe um objeto da classe Habilidade como parametro e constroi *skill* usando os valores do objeto passado como parametro.

virtual int Boss::type ()[override], [virtual] - O metodo que retorna 1 para o tipo Boss.

Referência da Classe Habilidade

#include “Habilidade.h”

Membros Públicos

Habilidade (std::string, int, int)
Habilidade ()
int get_damage ()
int get_spend ()
std::string **get_name** ()
void **set_nome** (std::string)
void **set_damage** (int)
void **set_spend** (int)

Descrição detalhada

A classe tem como atributos int *damage* - dano dado ao usar a habilidade - e int *spend* - o gasto de stamina ao usar a habilidade - e string *name* . Tem como metodos setters e getters dos atributos. Habilidades são passadas para Personagem e Boss para serem usados nas batalhas.

Construtores e Destrutores

Habilidade::Habilidade (std::string , int , int)

Habilidade::Habilidade ()

Funções membros

int Habilidade::get_damage ()

std::string Habilidade::get_name ()

int Habilidade::get_spend ()

void Habilidade::set_damage (int)

void Habilidade::set_nome (std::string)

void Habilidade::set_spend (int)

Referência da Classe Inventario

#include <Inventario.h>

Membros Públicos

Inventario ()
int get_gold ()
void **set_gold** (int num)
void add_armor (Armadura A)
void add_weapon (Arma A)
int armor_inventory_position (int id)
int weapon_inventory_position (int id)
void **remove_armor** (int id)
void **remove_weapon** (int id)
std::vector< Armadura > get_full_unused_armor ()
std::vector< Arma > get_full_unused_weapon ()
void display_inventory ()

Descrição detalhada

A classe Inventário foi feita para reaproveitar os itens na mecânica de compra e venda da loja. Ela tem 3 atributos:
1 inteiro para gold; 1 vector do atributo Arma(weapon); 1 vector do atributo Armadura (armor);

Construtores e Destrutores

Inventario::Inventario () - constrói o Inventario com o atributo gold como um inteiro de valor 20 por default.

Funções membros

void Inventario::add_armor (Armadura *A*) - adiciona uma armadura ao atributo armor;

void Inventario::add_weapon (Arma *A*) - adiciona uma arma ao atributo weapon;

int Inventario::armor_inventory_position (int *id*) - retorna a posição no vector de Armadura do Inventario com id correspondente.

void Inventario::display_inventory () - imprime os vetores armor e weapon, objeto a objeto

std::vector<Armadura> Inventario::get_full_unused_armor () - retorna um vector do tipo Armadura igual ao atributo armor;

std::vector<Arma> Inventario::get_full_unused_weapon () - returna um vector do tipo Arma igual ao atributo weapon;

int Inventario::get_gold ()

void Inventario::remove_armor (int *id*) - remove a armadura de atributo id igual ao parametro id;

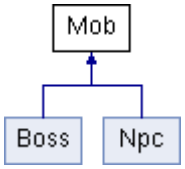
void Inventario::remove_weapon (int *id*) - remove a arma de atributo id igual ao parametro id;

void Inventario::set_gold (int *num*)

int Inventario::weapon_inventory_position (int *id*) - retorna a posição no vector de Arma do Inventario com id correspondente.

Referência da Classe Mob

#include <mob.h>
Diagrama de hierarquia para Mob:



Membros Públicos

virtual float **get_max_attack** ()=0
virtual float **get_min_attack** ()=0
virtual float **get_defense** ()=0
virtual std::string **get_name** ()=0
virtual float **get_life** ()=0
virtual void **set_max_attack** (float)=0
virtual void **set_min_attack** (float)=0
virtual void **set_defense** (float)=0
virtual void **set_life** (float)=0
virtual int **type** ()=0

Descrição detalhada

A classe é uma interface para os mobs do jogo, com metodos virtual getters e setters para ataque maximo, ataque minimo, vida, defesa e nome. Os Mobs são usados na Função Batalha e Função Treino.

Funções membros

virtual float Mob::get_defense ()[pure virtual]

Implementado por **Boss** (*p.9*) e **Npc** (*p.18*).

virtual float Mob::get_life ()[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual float Mob::get_max_attack ()[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual float Mob::get_min_attack ()[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual std::string Mob::get_name ()[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual void Mob::set_defense (float)[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual void Mob::set_life (float)[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual void Mob::set_max_attack (float)[pure virtual]

Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual void Mob::set_min_attack (float)[pure virtual]

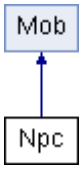
Implementado por **Boss** (*p.10*) e **Npc** (*p.18*).

virtual int Mob::type ()[pure virtual]- retorna um valor inteiro correspondente ao tipo da sub-classe instanciada.

Implementado por **Boss** (*p.**Erro! Indicador não definido.***) e **Npc** (*p.18*).

Referência da Classe Npc

#include <npc.h>
Diagrama de hierarquia para Npc:



Membros Públicos

Npc (std::string, float, float, float, float)
Npc (std::string)
Npc ()
virtual float **get_max_attack** () override
virtual float **get_min_attack** () override
virtual float **get_defense** () override
virtual std::string **get_name** () override
virtual float **get_life** () override
virtual void **set_max_attack** (float) override
virtual void **set_min_attack** (float) override
virtual void **set_defense** (float) override
virtual void **set_life** (float) override
void **set_name** (std::string)
virtual int **type** () override

Descrição detalhada

A classe é uma sub-classe de Mob, tem como atributos *float* life, max_attack, min_attack, defense e uma *string* para name. Possui metodos setters e getters para todos os atributos e implementa todos os metodos de Mob. Possui um construtor vazio e um que recebe todos uma *string* e 4 valores *float*.

Construtores e Destrutores

Npc::Npc (std::string , float , float , float , float)

Npc::Npc (std::string)

Npc::Npc ()

Funções membros

virtual float Npc::get_defense ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual float Npc::get_life ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual float Npc::get_max_attack ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual float Npc::get_min_attack ()[override], [virtual]

Implementa **Mob** (*p.15*).

virtual std::string Npc::get_name ()[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Npc::set_defense (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Npc::set_life (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Npc::set_max_attack (float)[override], [virtual]

Implementa **Mob** (*p.16*).

virtual void Npc::set_min_attack (float)[override], [virtual]

Implementa **Mob** (*p.16*).

void Npc::set_name (std::string)

virtual int Npc::type ()[override], [virtual]

Implementa **Mob** (*p.16*).

Referência da Classe Personagem

#include <Personagem.h>

Membros Públicos

Personagem (std::string _name)
Personagem (std::string _name, int _life, int _defense, int _attack, int _stamina)
std::string **get_name** ()
int get_life ()
void **set_life** (int life_change)
int get_defense ()
void **set_defense** (int def_change)
int get_attack ()
void **set_attack** (int attack_change)
int get_stamina ()
void **set_stamina** (int stamina_change)
int get_gold ()
void **set_gold** (int num)
void display_inventory ()
bool **check_gold** (int price)
std::vector< Armadura > get_inventory_armor ()
std::vector< Arma > get_inventory_weapon ()
int armor_inventory_position (int id)
int weapon_inventory_position (int id)
std::string **get_armor_name** ()
int get_armor_defense ()
int get_armor_id ()
int get_armor_price ()
void display_armor ()
std::string **get_weapon_name** ()
int get_weapon_attack ()
int get_weapon_id ()
int get_weapon_price ()
void display_weapon ()
void add_skill (Habilidade hab)
void **remove_skill** (int y)
void display_skill ()
int get_nskill ()
Habilidade get_skill (int)
void add_armor (Armadura toequip)
void **add_weapon** (**Arma** toequip)
void **remove_armor** (int id)
void **remove_weapon** (int id)
void equip_armor (Armadura toequip)
void unequip_armor ()
void **equip_weapon** (**Arma** toequip)
void unequip_weapon ()

void display_stats ()

Descrição detalhada

A classe foi criada para criar o personagem principal do jogo. Ela possui um 9 atributos, sendo: 1 string para o name; 4 inteiros para life, defense, attack, stamina; 3 atributos(objetos) de classes próprias Inventario, Armadura e Arma; 1 vector de classe própria Habilidade;

Construtores e Destrutores

Personagem::Personagem (std::string _*name*) - constrói os atributos name com a string recebida e life, defense, attack e stamina em 100, 20, 20 e 100, respectivamente, por default.

Personagem::Personagem (std::string _*name*, int _*life*, int _*defense*, int _*attack*, int _*stamina*) - constrói os atributos name, life, defense, attack e stamina com os atributos recebidos.

Funções membros

void Personagem::add_armor (Armadura *toequip*)

void Personagem::add_skill (Habilidade *hab*)

void Personagem::add_weapon (Arma *toequip*)

int Personagem::armor_inventory_position (int *id*)

bool Personagem::check_gold (int *price*)

void Personagem::display_armor ()

void Personagem::display_inventory ()

void Personagem::display_skill ()

void Personagem::display_stats ()

void Personagem::display_weapon ()

void Personagem::equip_armor (Armadura *toequip*)

void Personagem::equip_weapon (Arma *toequip*)

int Personagem::get_armor_defense ()

int Personagem::get_armor_id ()

std::string Personagem::get_armor_name ()

int Personagem::get_armor_price ()

int Personagem::get_attack ()

int Personagem::get_defense ()

int Personagem::get_gold ()

std::vector<Armadura> Personagem::get_inventory_armor ()

std::vector<Arma> Personagem::get_inventory_weapon ()

int Personagem::get_life ()

std::string Personagem::get_name ()

int Personagem::get_nskill ()

Habilidade Personagem::get_skill (int)

int Personagem::get_stamina ()

int Personagem::get_weapon_attack ()

int Personagem::get_weapon_id ()

std::string Personagem::get_weapon_name ()

int Personagem::get_weapon_price ()

void Personagem::remove_armor (int *id*)

void Personagem::remove_skill (int *y*)

void Personagem::remove_weapon (int *id*)

void Personagem::set_attack (int *attack_change*)

void Personagem::set_defense (int *def_change*)

void Personagem::set_gold (int *num*)

void Personagem::set_life (int *life_change*)

void Personagem::set_stamina (int *stamina_change*)

void Personagem::unequip_armor ()

void Personagem::unequip_weapon ()

int Personagem::weapon_inventory_position (int *id*)

Arquivos

Referência do Arquivo Arma.h

#include <string>

Componentes

class **Arma**

Referência do Arquivo Armadura.h

#include <string>

Componentes

class Armadura

Referência do Arquivo Batalha.h

#include "Personagem.h"
#include "mob.h"
#include "boss.h"

Funções

int **batalha** (**Personagem** &heroi, **Mob** &npc)
void **menu2** (**Personagem** &heroi, **Mob** &npc)
int **true_damage** (int ataque, double defesa)
void **atacar** (**Personagem** &heroi, **Mob** &5npc)
void atacar_mob (Personagem &heroi, Mob &npc)
int atacar_hab (Personagem &heroi, Mob &npc, Habilidade hab)
void atacar_hab_boss (Personagem &, Boss &)
void **fugir** (**Personagem** &heroi)

Funções

void atacar (Personagem & *heroi*, Mob & *npc*) - A função recebe dois parâmetros, um “Personagem” (protagonista) e um “Mob” (inimigo), e é responsável por realizar o ataque do herói no oponente. Para isto, o procedimento busca o valor de ataque do personagem, a defesa do Mob, e chama a função “true_damage” para calcular o dano real. Após este passo, o método desconta da vida do oponente o valor que foi retornado desta função

int atacar_hab (Personagem & *heroi*, Mob & *npc*, Habilidade *hab*) - A função recebe parametros de 3 classes (Personagem, Mob e Habilidade), primeiro testa se o Personagem tem stamina suficiente para usar a habilidade passada e o valor de retorno da função é baseado nisso. Caso tenha stamina a função chama *true_damage()* e desconta o dano da vida ddo Npc e a stamina do Personagem.

void atacar_hab_boss (Personagem & , Boss &) - A função recebe Personagem que sera atacado e o Boss que será o atacante. A função chama true_damage() para calcular o dano e desconta da vida do Personagem.

void atacar_mob (Personagem & *heroi*, Mob & *npc*) – A função recebe dois parâmetros, um “Personagem” (protagonista) e um “Mob” (inimigo), e é responsável por realizar o ataque do oponente no herói. Para isto, o procedimento busca o valor de ataque do Mob, a defesa do personagem, e chama a função “true_damage” para calcular o dano real. Após este passo, o método desconta da vida do protagonista o valor que foi retornado desta função.

int batalha (Personagem & *heroi*, Mob & *npc*) - A função batalha recebe dois parâmetros, um “Personagem” (protagonista) e um “Mob” (inimigo) e é a responsável pelo confronto entre o personagem e o inimigo. Nela, o protagonista recebe e inflige dano, podendo derrotar o oponente e receber uma recompensa, ser derrotado e perder o jogo, ou fugir, pagando uma certa quantia pela misericórdia. A função retorna um valor inteiro entre 0 e 2, informando o resultado da partida (0 = perdeu, 1 = ganhou, 2 = fugiu).

void fugir (Personagem & *heroi*) – A função, quando chamada, abandona da batalha.

void menu2 (Personagem & *heroi*, Mob & *npc*) - A função é dedicada a parte de habilidades na batalha. A função recebe um Personagem e um Mob, mostra as habilidades do Personagem com o metodo *display_skill(),* pede a entrada da habilidade e chama a função *atacar_hab()* com a opção selecionada. A função testa se *atacar_hab()* vai funcionar e então testa se o Mob passado é um Boss, caso seja, é chamado a função *atacar_hab_boss()* e caso seja um Npc é chamada a função *atacar_mob().*

int true_damage (int *ataque*, double *defesa*) - A função true_damage recebe dois parâmetros, um inteiro (ataque) e um double (defesa). Ela é responsável por calcular o dano real de um ataque, seja do protagonista ou do inimigo. Em outras palavras, ela desconta uma porcentagem do dano de ataque a partir dos pontos de defesa de quem o recebe multiplicado por uma taxa constante ($4\sqrt{x}$) e retorna o resultado.

Referência do Arquivo Boss.h

#include "mob.h"
#include "Habilidade.h"
#include <string>

Componentes

class **Boss**

Referência do Arquivo FunçãoLoja.h

#include <vector>
#include <fstream>
#include "Personagem.h"
#include "Arma.h"
#include "Armadura.h"

Funções

void present_store ()
void menu_store ()
void seller_random_fail_speech ()
void confirmation_checkout ()
void instruction_store (Personagem A)
void remaining_gold (Personagem A)
void **reading_file** (std::ifstream &file, std::vector< std::string > &archives_name, std::vector< int > &archives_id, std::vector< int > &archives_price, std::vector< int > &archives_attribute)
std::vector< Arma > create_objects_weapon ()
std::vector< Armadura > create_objects_armor ()
void **buy_weapon** (**Personagem** &A, std::vector< **Arma** > &weapons)
void **buy_armor** (**Personagem** &A, std::vector< **Armadura** > &armors)
void **buy_potions** (**Personagem** &A, int price_on_potions)
void **sell_armor** (**Personagem** &A, std::vector< **Armadura** > inventory_armor)
void **sell_weapon** (**Personagem** &A, std::vector< **Arma** > inventory_weapon)
void store_weapon (Personagem &A)
void store_armor (Personagem &A)
void store_potions (Personagem &A)
void store_inventory (Personagem &A)
void Funcao_Loja (Personagem &A)

Funções

void buy_armor (Personagem & *A*, std::vector< Armadura > & *armors*): Recebe um Personagem A e um vector do tipo Armadura armors, ambos por referência. Recebe um número inteiro. Compara com os ids dos objetos do vector recebido. Se o número digitado for 0, a função é finalizada. Verifica a existência e o *price* do objeto a ser comprado no vector. Se a compra for possível (atributo *gold* do *inventory* do Personagem recebido >= *price* do objeto), é descontada a quantidade de gold e equipado o item no Personagem, e é impressa uma mensagem de sucesso e o novo valor do atributo *defense* do Personagem. Se a compra não for possível, imprime uma mensagem de erro e a função é chamada novamente com os mesmos parâmetros.

void buy_potions (Personagem & *A*, int *price_on_potions*): Recebe um Personagem A por referência e um inteiro price_on_potion. Recebe um número inteiro. Se o número digitado for 0, a função é finalizada. Se o número digitado for 1, 2, 3 ou 4, a função checa se é possível comprar o item

digitado (padrões pré-carregados para números entre 1 e 4). Se a compra for possível (atributo *gold* do *inventory* do Personagem recebido >= *preço* pré estabelecido), é descontado o gold e são utilizados métodos set_life(quantidade) ou set_stamina(quantidade), com a quantidade pré-estabelecida para cada item. Se a compra não for possível

void buy_weapon (Personagem & *A*, std::vector< Arma > & *weapons*): Recebe um Personagem A e um vector do tipo Arma armors, ambos por referência. Recebe um número inteiro. Compara com os ids dos objetos do vector recebido. Se o número digitado for 0, a função é finalizada. Verifica a existência e o *price* do objeto a ser comprado no vector. Se a compra for possível (atributo *gold* do *inventory* do Personagem recebido >= *price* do objeto), é descontada a quantidade de gold e equipado o item no Personagem, e é impressa uma mensagem de sucesso e o novo valor do atributo *attack* do Personagem. Se a compra não for possível, imprime uma mensagem de erro e a função é chamada novamente com os mesmos parâmetros.

void confirmation_checkout (): Imprime o menu de confirmação de venda de item.

std::vector<Armadura> create_objects_armor (): Tenta abrir um arquivo com texto de criação das armaduras (parâmetros do construtor) e cria um vector do tipo Armadura. Se o arquivo for aberto e lido de forma correta, retorna o vector do tipo Armadura com tamanho igual a numero de linhas do arquivo. Se o arquivo não for aberto, retorna um vector vazio do tipo Arma.

std::vector<Arma> create_objects_weapon (): Tenta abrir um arquivo com texto de criação das armas (parâmetros do construtor) e cria um vector do tipo Arma. Se o arquivo for aberto e lido de forma correta, retorna o vector do tipo Arma com tamanho igual a numero de linhas do arquivo. Se o arquivo não for aberto, retorna um vector vazio do tipo Arma.

void Funcao_Loja (Personagem & *A*): Recebe um personagem por referência. Imprime funções de texto e recebe um número inteiro. Se o número inteiro for 5, a função é finalizada. Para cada outro número válido (1, 2, 3 ,4), chama funções de impressão de texto, compra, venda ou confirmação pré-estabelecidas acima
void instruction_store (Personagem *A*): Imprime o texto de instruções para compra de itens da loja e imprime o número de moedas(gold) do personagem recebido.

void menu_store (): Imprime o menu principal da loja.

void present_store (): Imprime o texto de apresentação da loja.

void reading_file (std::ifstream & ***file*****, std::vector< std::string > &** ***archives_name*****, std::vector< int > &** ***archives_id*****, std::vector< int > &** ***archives_price*****, std::vector< int > &** ***archives_attribute*****):** Recebe um arquivo ".txt" aberto por referência, e 4 vectors (tipos string, int, int, int, respectivamente) por referência. Lê o arquivo linha por linha, lê a linha até o delimitador ',' e copia o valor da variável recebida para a posição [número da linha - 1] nos vectors recebidos, na ordem string, int, int, int

void remaining_gold (Personagem ***A*****):** Imprime uma fala do Vendedor com o número de moedas(gold) do personagem recebido

void sell_armor (Personagem & ***A*****, std::vector< Armadura >** ***inventory_armor*****):** Recebe um personagem por referência e carrega os itens do vetor armor de seu *inventory*.Recebe um número inteiro e compara o número recebido com os ids do vetor armor. Se o número digitado for 0, a função é finalizada. Se o número corresponde ao id de algum item de armor, é pedida uma confirmação de venda. Se for confirmada, o gold equivalente é adicionado ao inventory do Personagem e o item é excluido de armors. Se não for confirmada, é impressa uma fala de erro do Vendedor e a função é finalizada.

void sell_weapon (Personagem & ***A*****, std::vector< Arma >** ***inventory_weapon*****):** Recebe um personagem por referência e carrega os itens do vetor weapon de seu *inventory*. Recebe um número inteiro e compara o número recebido com os ids do vetor weapon. Se o número digitado for 0, a função é finalizada. Se o número corresponde ao id de algum item de weapon, é pedida uma confirmação de venda. Se for confirmada, o gold equivalente é adicionado ao inventory do Personagem e o item é excluido de weapon. Se não for confirmada, é impressa uma fala de erro do Vendedor e a função é finalizada

void seller_random_fail_speech (): Escolhe e imprime uma fala de erro do vendedor

void store_armor (Personagem & ***A*****):** Recebe um personagem por referência. Cria um vetor com objetos do tipo Armadura pela função create_objects_armor e imprime os atributos de cada objeto criado. Chama funções de texto e confirmação de compra pela função buy_armor

void store_inventory (Personagem & ***A*****):** Recebe um personagem por referência**.** Imprime menus de venda de itens do Personagem. Recebe um

número inteiro. Se o número digitado for 0, é impressa uma mensagem do Vendedor e a função é finalizada. Se o número digitado for válido (diferente de 0 e menor que 2): (1) Checa se o tamanho do vector armor é maior que 0, se sim imprime os objetos do vector armor do Personagem, chama funções de texto e confirmação de compra pela função sell_armor. Se não, imprime uma mensagem de erro e a função é finalizada;

(2) Checa se o tamanho do vector weapon é maior que 0, se sim imprime os objetos do vector weapon do Personagem, chama funções de texto e confirmação de compra pela função sell_weapon. Se não, imprime uma mensagem de erro e a função é finalizada

void store_potions (Personagem & *A*): Imprime itens pré-estabelecidos (poções) e estabelece o preço delas. Chama funções de texto e confirmação de compra pela função buy_potion

void store_weapon (Personagem & *A*): Recebe um personagem por referência. Cria um vetor com objetos do tipo Arma pela função create_objects_weapon e imprime os atributos de cada objeto criado. Chama funções de texto e confirmação de compra pela função buy_weapon

Funções Gerais

Definições e Macros

#define **n_enemies** 3

Funções

void **pausar** ()
bool exit_game ()
void **quiz** (int n, **Personagem** &player)
void **add_skill_pers** (**Personagem** &player, int fase)
void **add_skill_boss** (**Boss** &boss, int fase)
void **checker** (int *variavel, int p1, int p2)
std::string **adjust_square** (std::string to_print, int num)
void **print_square** (std::string to_print1, std::string to_print2, std::string to_print3, std::string to_print4)
void **exit** ()
int **reward** (int fase_num, int multiplicador, **Personagem** &player)

Definições e macros

#define n_enemies 3

Funções diversas que podem ser utilizadas em todas as classes

---

Funções

void add_skill_boss (Boss & *boss*, int *fase*)

void add_skill_pers (Personagem & *player*, int *fase*)

std::string adjust_square (std::string *to_print*, int *num*)

void checker (int * *variavel*, int *p1*, int *p2*) - verifica o tipo de entrada que o usuário insere e checar (por isso o nome “Checker”) se a entrada é válida. A função recebe uma variável como ponteiro e duas variáveis que são os limitadores. Dentro da função armazenamos o valor inserido pelo usuário e utilizamos as ferramentas Try/Throw/Catch para receber a variável, testar sua validade e tratá-la. Caso a entrada seja válida, a função retorna o valor inserido pelo usuário. Caso a entrada seja inválida, a função solicita ao usuário que insira uma entrada válida.

bool exit_game () – abandona o jogo quando chamada.

void pausar () – pausa o jogo quando chamada.

void print_square (std::string *to_print1*, std::string *to_print2*, std::string *to_print3*, std::string *to_print4*) - cria e imprime uma tabela simétrica com 35 caracteres de espaço até a borda da tabela.

void quiz (int *n*, Personagem & *player*) – armazena os quiz chamados no decorrer do jogo e, caso o jogador acerte o quiz, ele recebe ouro como recompensa.

int reward (int *fase_num*, int *multiplicador*, Personagem & *player*) - define o ouro que o Npc dropará por meio de um multiplicador de recompensa (por isso o nome “Reward) de acordo com a fase do jogo .A função recebe o numero da fase do jogo e o multiplicador que varia de acordo com a fase. Cada vez que a função é chamada, ela cria um número que varia entre 10 e 20. A fórmula da recompensa é dada por
(multiplicador ^(fase do jogo – 1))*número que varia entre 10 e 20 e ela retorna um número inteiro

Referência do Arquivo Habilidade.h

#include <string>

Componentes

class Habilidade

Referência do Arquivo Inventario.h

#include "Arma.h"
#include "Armadura.h"
#include <vector>

Componentes

class Inventario

Referência do Arquivo jogo.h

```
#include <iostream>
#include <stdio.h>
#include "Habilidade.h"
#include "Batalha.h"
#include "npc.h"
#include "boss.h"
#include "Personagem.h"
#include "FuncoesGerais.h"
#include "menu.h"
```

Definições e Macros

#define **n_enemies** 3

Funções

void **Fase_1** (**Personagem** &player, int t_num)
void **Fase_2** (**Personagem** &player, int t_num)
void **Fase_3** (**Personagem** &player, int t_num)
void **Fase_4** (**Personagem** &player, int t_num)
void **Fase_5** (**Personagem** &player, int t_num)
void **Fase_6** (**Personagem** &player, int t_num)
void **Fase_7** (**Personagem** &player, int t_num)

Definições e macros

#define n_enemies 3

A função é responsável por construir o Personagem (protagonista) usando o construtor que recebe o parâmetro “nome” (string), informado pelo usuário. Após isso, chama as fases..

Funções

void Fase_1 (Personagem & *player*, int *t_num*)

void Fase_2 (Personagem & *player*, int *t_num*)

void Fase_3 (Personagem & *player*, int *t_num*)

void Fase_4 (Personagem & *player*, int *t_num*)

void Fase_5 (Personagem & *player*, int *t_num*)

void Fase_6 (Personagem & *player*, int *t_num*)

void Fase_7 (Personagem & *player*, int *t_num*)

- As fases (1 a 7) tem funcionamento semelhante entre elas, diferenciando apenas a história e o parâmetro “t_num”. Cada função recebe dois parâmetros: “Personagem” (herói) e “t_num” (número máximo de treinos permitidos na fase). Ao início da função, são construídos os inimigos e ao longo da fase a função chama os métodos responsáveis pela batalha, treino, loja, e desenvolve a história.

Referência do Arquivo Menu.h

```
#include "FuncoesGerais.h"
#include "FuncaoLoja.h"
#include "Personagem.h"
#include "jogo.h"
#include "npc.h"
#include "treino.h"
```

Funções

void show_init ()
void **show_menu** (**Personagem** &heroi, **Npc** &npc, int t_num, int fase_num)
void show_menu_h ()

Funções

void show_init () – exibe o texto ao iniciar o jogo.

void show_menu (Personagem & ***heroi*****, Npc &** ***npc*****, int** ***t_num*****, int** ***fase_num*****)** – Essa função é responsável pelo menu entre as fases. Dentro dela, chama-se outros métodos de desenvolvimento do jogo, tais como a função treinar, função loja e exit game. Ela tem como parâmetro objetos da classe Personagem e Npc porque nas funções adjacentes esses parâmetros são necessários. Já t_num e fase_num são inteiros referentes ao número de treino disponível para o jogador e o número da fase que o jogador se encontra, respectivamente.

void show_menu_h () – Essa função é exibida após a história/diálogo exibida com o passar do jogo. Ela pergunta ao jogador se ele deseja continuar ou abandonar o jogo.

Referência do Arquivo mob.h

#include <string>

Componentes

class **Mob**

Referência do Arquivo Npc.h

#include "mob.h"
#include <string>

Componentes

class **Npc**

Referência do Arquivo Personagem.h

```
#include <string>
#include <vector>
#include "Inventario.h"
#include "Armadura.h"
#include "Arma.h"
#include "Habilidade.h"
```

Componentes

class Personagem

Referência do Arquivo treino.h

```
#include <iostream>
#include <time.h>
#include <math.h>
#include "treino.h"
#include "Batalha.h"
#include "FuncoesGerais.h"
#include "Personagem.h"
#include "npc.h"
```

Funções

void **treinar** (**Personagem** &heroi, **Npc** &npc, int t_num, int fase_num): O intuito deste método convocar batalhas fora da história principal para que o protagonista possa recolher moedas para melhorar seus equipamentos. Para isso, ele confere se a quantidade de treinos não ultrapassou o máximo permitido para a fase em que ele se encontra e, se este número ainda não foi antingido, ele aciona a função batalha para que o confronto ocorra contra o Npc que foi recebido como parâmetro. Após isso, este procedimento confere o resultado da luta e, a partir daí, realiza a conclusão necessária (seja recompensar, punir ou terminar o jogo).