Documentazione Tecnica
Progetto Basi di Dati
DungeonAsDB - Comelico Simulator 2017

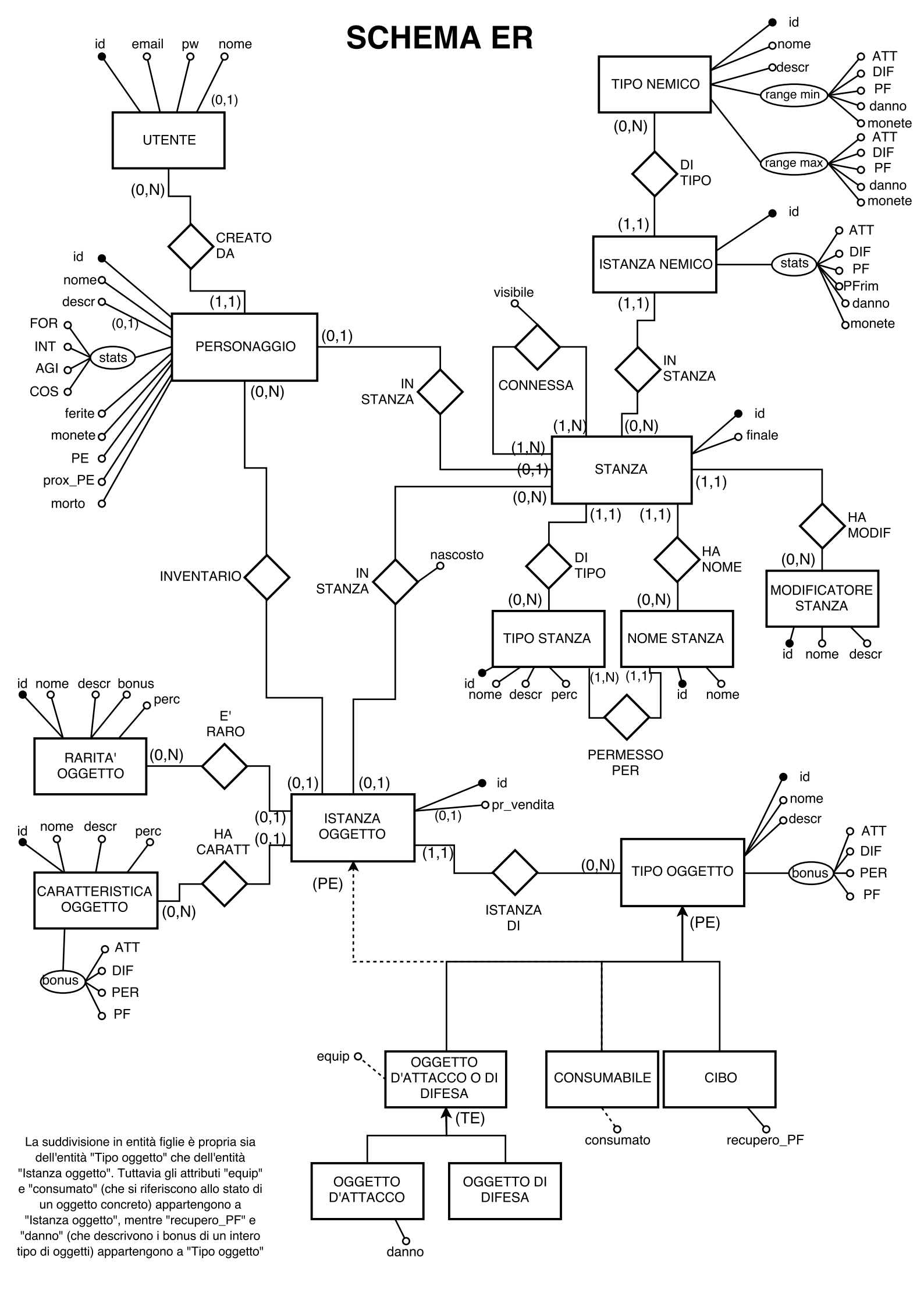
Elvis Nava 870234 - Dario Ostuni 870321

Gennaio 2017

Comelico Simulator 2017 è un'applicazione web che consente a utenti registrati di partecipare a un gioco di ruolo di genere *survival roguelike*, in cui si possono creare personaggi e con essi esplorare sedi universitarie abbandonate generate casualmente, combattere nemici di difficoltà crescente, trovare oggetti di diverse rarità e variazioni, e scambiarli con altri giocatori per valuta di gioco. I dati di gioco sono memorizzati in una base di dati e la logica di gioco è implementata completamente con procedure e trigger nel linguaggio procedurale della base di dati stessa.

Per l'implementazione sono stati scelti PostgreSQL come DBMS, PL/Python 3 come linguaggio procedurale e python come linguaggio per lo sviluppo dell'interfaccia web.

SCHEMA ER



La suddivisione in entità figlie è propria sia dell'entità "Tipo oggetto" che dell'entità "Istanza oggetto". Tuttavia gli attributi "equip" e "consumato" (che si riferiscono allo stato di un oggetto concreto) appartengono a "Istanza oggetto", mentre "recupero_PF" e "danno" (che descrivono i bonus di un intero tipo di oggetti) appartengono a "Tipo oggetto"

1 Schema Concettuale ER della base di dati

Concettualmente il Database è costituito dalle entità principali:

1.1 Utente

Rappresenta l'utente registrato all'applicazione, ha come attributi un id, i dati di login e un nome opzionale. È in grado di creare più Personaggi (relazione uno a molti).

1.2 Personaggio

Al centro dell'esperienza di gioco, è in relazione con diverse entità con cui può interagire. È creato da un Utente (relazione uno a molti), può trovarsi in una Stanza se la partita è cominciata (relazione uno a uno), ha un inventario di Oggetti (relazione uno a molti). Come attributi ha id, nome, descrizione opzionale, attributi base di gioco (FOR, INT, AGI, COS), contatore di ferite, di monete, Punti Esperienza (PE), PE accumulati nella partita corrente, e un flag per indicarne la morte. Tutti gli altri dati necessari per la gestione del Personaggio sono direttamente derivabili da questi o da attributi di entità con esso in relazione.

1.3 Stanza

Componente base per costruire dungeon, può ospitare un Personaggio (relazione uno a uno), può contenere Oggetti anche nascosti (relazione uno a molti), può contenere Nemici (relazione uno a molti), può essere connessa con un'altra Stanza attraverso un passaggio visibile o meno (relazione molti a molti). Come attributi propri dell'entità vi sono solo l'id e un flag per indicare se la stanza è finale (ovvero è possibile terminare l'avventura trovandosi in essa). Per il resto le differenti varietà di stanze generabili sono caratterizzate dalle relazioni (uno a molti) con Tipo Stanza per il nome e una descrizione generici, Nome Stanza per il nome proprio, e Modificatore Stanza per una descrizione aggiuntiva. Il Tipo Stanza ha anche un attributo contenente la percentuale di probabilità che sia scelto. Inoltre un Nome Stanza è permesso solamente per un certo Tipo (relazione uno a molti).

1.4 Istanza e Tipo Nemico

L'Istanza Nemico rappresenta un nemico concreto presente in una partita in corso, si trova in una Stanza (relazione uno a molti) e prende nome e descrizione dal suo Tipo (relazione uno a molti). I suoi attributi sono un id, i valori degli attributi di gioco con cui è stato generato (ATT, DIF, PF, danno), i Punti Ferita rimanenti (PFrim) e le monete che farà cadere alla morte.

Il Tipo Nemico rappresenta astrattamente un nemico possibilmente generabile dal gioco. I suoi attributi sono un id, un nome, una descrizione, il range per i valori possibili di ogni attributo di gioco del nemico (ATT, DIF, PF, danno) e per le monete fatte cadere alla morte.

1.5 Istanza e Tipo Oggetto

L'Istanza Oggetto rappresenta un oggetto concreto presente in un dungeon o nelle mani di un Personaggio. Può infatti trovarsi in una Stanza (relazione uno a molti) o alternativamente nell'inventario di un Personaggio (relazione uno a molti). Come attributi propri dell'entità vi sono solo l'id e un eventuale prezzo di vendita, nel caso l'oggetto sia stato messo in vendita nel mercato di gioco. Le diverse varietà di oggetti generabili sono caratterizzate dalle relazioni (uno a molti) con Tipo Oggetto per ottenere nome, descrizione e i bonus di base agli attributi (ATT, DIF, PER, PF), e facoltativamente con Caratteristica e/o Rarità Oggetto per ulteriori bonus. Caratteristica e Rarità Oggetto hanno ciascuna come attributi un nome, una descrizione e la percentuale di probabilità di essere selezionate durante la generazione. Caratteristica Oggetto ha anche valori dei bonus aggiuntivi differenti per ogni attributo, mentre Rarità Oggetto ha un bonus aggiuntivo unico per tutti gli attributi di gioco.

Tipo Oggetto, che rappresenta astrattamente un oggetto generabile dal gioco, fa inoltre parte di una gerarchia di generalizzazione Parziale Esclusiva con le entità figlie Consumabile, Cibo, e Oggetto d'Attacco/Difesa, a sua volta suddivisa con una gerarchia Totale Esclusiva nelle entità Oggetto d'Attacco e Oggetto di Difesa. Questa suddivisione è in realtà propria sia di Tipo Oggetto che di Istanza Oggetto: infatti i Consumabili hanno come attributo un flag per indicare se l'oggetto è stato consumato e gli Oggetti di Attacco/Difesa un flag per indicare se è stato equipaggiato, entrambi attributi di Istanze concrete di Oggetti, mentre il Cibo e gli Oggetti d'Attacco hanno come attributi rispettivamente il recupero di Punti Ferita e il danno inferto, entrambi attributi di Tipi astratti di Oggetti.

In particolare occorre notare come l'entità Cibo non sia stata progettata come sottoclasse di Consumabile. Infatti gli Oggetti dei due tipi funzionano in modo differente: mentre un'istanza di Cibo è cancellata al momento del consumo e il suo valore di recupero PF viene usato immediatamente per alleviare ferite, un'istanza di Consumabile è flaggata e viene cancellata solo dopo uno spostamento di stanza: il flag attiva i suoi valori di bonus ma allo stesso tempo nasconde la continuata presenza dell'oggetto al giocatore.

2 Schema Relazionale della base di dati

Lo schema ER è stato tradotto nel modello relazionale con le seguenti relazioni:

UTENTE(id, email, pw, nome*)

PERSONAGGIO(id, nome, descr*, _FOR, _INT, _AGI, _COS, monete, PE, prox_PE, morto, *creato_da*, *in_stanza**)

STANZA(id, finale, *tipo*, *nome*, *modif*)

TIPO_STANZA(id, nome, descr, perc)

NOME_STANZA(id, nome, *permesso_per*)

MODIF_STANZA(id, nome, descr)

CONNESSA(*stanza1*, *stanza2*, visibile)

TIPO_NEMICO(id, nome, descr, min_ATT, max_ATT, min_DIF, max_DIF, min_PF, max_PF, min_danno, max_danno, min_monete, max_monete)

IST_NEMICO(id, _ATT, _DIF, _PFmax, _PFrim, _danno, monete, *in_stanza*, *tipo*)

TIPO_OGGETTO(id, nome, _ATT, _DIF, _PER, _PF, _danno*, classe*, recupero_PF*)

CAR_OGGETTO(id, nome, descr, add_b_ATT, add_b_DIF, add_b_PER, add_b_PF, perc)

RARITA_OGGETTO(id, nome, descr, bonus, perc)

IST_OGGETTO(id, *di_personaggio*, *in_stanza*, nascosto*, pr_vendita*, equip*, consumato*, *istanza_di*, *car*, *rarita*)

2.1 Note sulla traduzione e sui controlli

Le relazioni uno a molti dello schema ER sono state tradotte con l'inserimento di una chiave esterna nella tabella corrispondente all'entità con cardinalità massima 1. La relazione molti a molti tra stanze è stata tradotta con una tabella (CONNESSA) avente come chiave primaria una coppia di chiavi esterne riferite agli id delle stanze, con aggiunto l'attributo di relazione "visibile".

La gerarchia Parziale Esclusiva tra le classi di oggetti è stata tradotta conservando solamente l'entità padre, inserendo un attributo "classe" in TIPO_OGGETTO che può assumere i valori: "_ATT" se l'oggetto è d'attacco, "_DIF" se di difesa (l'ulteriore suddivisione Totale Esclusiva è stata precedentemente collassata mantenendo le sottoclassi), "cons" se è un consumabile, "cibo" se è cibo, o NULL (essendo la relazione parziale).

In TIPO_OGGETTO si controlla con clausole CHECK che _danno non sia NULL se la classe è "_ATT" e sia NULL altrimenti, e che recupero_PF non sia NULL se la classe è "cibo" e sia NULL altrimenti. Altri attributi che hanno a che fare con la classe dell'oggetto (eqip e consumato) sono tuttavia parte di IST_OGGETTO, essendo proprietà di oggetti concreti. Il loro rispetto dei

vincoli di classe non può quindi essere mantenuto con clasuole CHECK, ma viene controllato da trigger.

In IST_OGGETTO sono inoltre specificati altri controlli: si controlla che almeno un attributo tra di_personaggio e in_stanza non sia NULL, e che non siano entrambi presenti contemporaneamente. Si controlla che pr_vendita possa non essere NULL solamente quando di_personaggio è specificato, e che nascosto possa non essere NULL solamente quando in_stanza è specificato.

Su UTENTE si effettua il controllo del campo email, per assicurarsi che matchi il pattern '%@%.%'.

2.2 Domini, Viste e tabelle aggiuntive

Per rappresentare certi dati sottoposti a vincoli sono stati aggiunti Domini aggiuntivi:

`base_bonus`: un valore intero tra -6 e +6 per i bonus di base (ATT, DIF, PER, PF, danno) dei tipi di oggetto.

`main_attribute`: un valore intero tra 3 e 18 per gli attributi di gioco base di un personaggio (FOR, INT, AGI, COS).

`classe_oggetto`: un valore di 4 caratteri compreso tra '_ATT', '_DIF', 'cons', e 'cibo'.

Sono state create diverse Viste per facilitare l'accesso ai dati:

`stanza_view`: effettua il join tra `tipo_stanza`, `nome_stanza` e `modif_stanza`, selezionando tutti gli attributi necessari per avere la descrizione completa di una stanza.

`ist_nemico_view`: effettua il join tra `ist_nemico` e `tipo_nemico` per recuperare il nome e la descrizione dell'istanza di un nemico (non seleziona i range per gli attributi, utili solo nella fase di generazione dell'istanza).

`ist_oggetto_view`: effettua il join tra `ist_oggetto`, `tipo_oggetto`, `car_oggetto` e `rarita_oggetto`, selezionando tutti gli attributi necessari per caratterizzare l'istanza di un oggetto. In particolare somma i bonus base (soggetti a limite -6/+6) con eventuali bonus aggiuntivi dati dalla caratteristica e dalla rarità dell'oggetto. Una vista aggiuntiva `ist_oggetto_view_no_nasc` seleziona solo gli oggetti non nascosti.

`personaggio_attr_deriv`: permette di ottenere gli attributi di gioco derivati (ATT, DIF, PER, PF massimi, PF rimanenti, danno) di un personaggio, calcolandoli a partire dai suoi attributi di gioco base e sommandoli ai bonus degli oggetti nel suo inventario (considerando oggetti di attacco o difesa solo se `equip`=true e oggetti consumati solo se `consumato`=true). I valori di bonus di inventario sono ottenuti con una subquery su `ist_oggetto_view` utilizzando l'operatore aggregato SUM sui bonus, raggruppando per personaggio.

Più tardi durante la progettazione dell'UI web è emersa la necessità di due tabelle aggiuntive:

SESSIONE(chiave, valore) per la memorizzazione delle sessioni degli utenti, avendo come "valore" un JSON contenente le variabili di python

LANCIO_DADI_ATTR(*utente*, roll1, roll2, roll3, roll4, roll5) per memorizzare temporaneamente i lanci di dadi di un utente prima della creazione di un personaggio. Occorre memorizzare i valori poichè devono essere visualizzati nell'UI per essere scelti.

3 Funzioni e Trigger

Come linguaggio procedurale per la realizzazione di funzioni e trigger è stato scelto `plpython3`.

3.1 Funzioni

3.1.1 `tira_dadi_attr(utente INTEGER) RETURNS void`

Tira 5 volte 3d6 e salva i valori ottenuti nella tabella di lanci temporanei `lancio_dadi_attr`. Accetta come parametro `utente` l'id dell'utente che effettua il lancio e ritorna `void`.

3.1.2 `crea_personaggio(nome TEXT, descr TEXT, rolli_for INTEGER, rolli_int INTEGER, rolli_agi INTEGER, rolli_cos INTEGER, utente INTEGER) RETURNS void`

Crea un nuovo personaggio utilizzando i dati forniti e i valori dei dadi della tabella `lancio_dadi_attr`. Il personaggio creato viene inserito in `personaggio`, vengono poi inseriti in `ist_oggetto` i due oggetti iniziali per l'avventura (un oggetto d'attacco e una razione di cibo), segnandoli come di proprietà del personaggio. Accetta come parametri `nome` e `descr` il nome e la descrizione del personaggio, in `rolli_for`, `rolli_int`, `rolli_agi`, `rolli_cos` gli indici da 0 a 4 dei dadi scelti per i valori degli attributi base di gioco, e in `utente` l'id dell'utente creatore del personaggio. Ritorna `void`.

3.1.3 `crea_grafo() RETURNS INTEGER`

Genera un grafo connesso di stanze per una nuova partita. Non prende parametri e ritorna l'id della stanza iniziale del dungeon.

Inizialmente vengono generate casualmente 32 stanze: viene deciso il `tipo_stanza` con un confronto tra `random()` e il valore `perc` di percentuale di probabilità che questo venga scelto, poi si selezionano casualmente un `nome_stanza` (tra quelli permessi per il tipo) e un `modif_stanza`, controllando infine di volta in volta che una stanza totalmente uguale non sia già stata generata. Le stanze create sono inserite una di seguito nella tabella `stanza`, facendo sì che i loro id siano in sequenza.

Successivamente viene utilizzato questo algoritmo per generare le connessioni e inserirle in `connessa`:

```
mat_adj = [[0 for j in range(32)] for i in range(32)]

for i in range(16):
    a, b = randrange(32), randrange(32)
    if a == b:
        continue
```

```
    mat_adj[a][b] = 2
    mat_adj[b][a] = 2
for i in range(1, 32):
    to_node = randrange(i)
    mat_adj[i][to_node] = 1
    mat_adj[to_node][i] = 1

for i in range(32):
    for j in range(32):
        if mat_adj[i][j] != 0:
            visibile = True
            if mat_adj[i][j] == 2:
                visibile = False
            plan = plpy.prepare("INSERT INTO connessa VALUES($1, $2,
                $3)", ["integer", "integer", "boolean"])
            plpy.execute(plan, [i+start_node, j+start_node, visibile])
```

L'algoritmo, rappresentando il grafo con una matrice di adiacenza `mat_adj`, genera prima alcuni passaggi casuali nascosti tra stanze (segnati nella matrice col valore 2), poi genera i passaggi restanti (segnati col valore 1) assicurandosi che il grafo sia connesso. Infine le connessioni sono inserite nella tabella, scorrendo la matrice e usando come id delle stanze gli indici i, j, sommati all'id della stanza iniziale (salvato precedentemente) `start_node`.

3.1.4 `inizia_partita(id_pers INTEGER) RETURNS void`

Inizia una nuova partita chiamando `crea_grafo()` per creare il dungeon, per poi popolarlo con oggetti e nemici di forza adatta al personaggio (prima controlla che il personaggio non si trovi già in una stanza e che non sia morto). Prende come parametro `id_pers` l'id del personaggio e ritorna `void`.

Prima di tutto viene ottenuto un valore `difficulty` dato dalla somma degli attributi derivati del personaggio (ATT, DIF, PER, PF), poi si esegue un ciclo sulle stanze del dungeon (dalla prima alla finale in ordine per id) ottenendo ogni volta un valore `room_difficulty` che parte da `difficulty` e aumenta fino a raddoppiare.

La presenza di nemici è decisa casualmente per tutte le stanze (tranne la prima e la finale) utilizzando una distribuzione geometrica di parametro 1/2 (ottenendo una media di 2 nemici per stanza). Un `tipo_nemico` è scelto di volta in volta con probabilità $\frac{1}{3}^{\frac{\texttt{difficulty\_nemico}}{\texttt{room\_difficulty}}}$, usando come `difficulty_nemico` la somma dei suoi attributi medi.

Gli oggetti sono generati in modo simile con una distribuzione geometrica di parametro 1/3 (ottenendo una media di 3 oggetti per stanza). Per la scelta del `tipo_oggetto` si effettua inizialmente un confronto tra `random()` e 0.3 per decidere se considerare i tipi di classe "cibo" o gli altri (così da alzare artificialmente il numero di oggetti "cibo" generati), poi si genera un determinato oggetto con probabilità $\frac{1}{3}^{\frac{5*\texttt{livello\_tipo}}{\texttt{room\_difficulty}}}$, usando come `livello_tipo` la somma dei bonus di base conferiti dal tipo di oggetto. Si scelgono a caso una `caratteristica_oggetto` e una `rarita_oggetto`, e si decide se includerle o meno nell'oggetto generato con un confronto di `random()` con il rispettivo valore del campo `perc`.

3.1.5 `finisci_partita(id_pers INTEGER) RETURNS void`

Controlla che il personaggio si trovi in una stanza finale, poi somma ai suoi `PE` (Punti Esperienza) i punti `prox_PE` accumulati durante la partita, lo toglie dalla stanza in cui si trova settando `in_stanza` a NULL e cancella le 32 stanze del dungeon all'indietro a partire da quella in cui si trova (gli id delle stanza erano tutti sequenziali). Accetta come parametro `id_pers` l'id del personaggio e ritorna void.

3.1.6 `mangia(id_personaggio INTEGER, id_ogg INTEGER) RETURNS void`

Accetta come parametro `id_personaggio` l'id del personaggio, come `id_ogg` l'id dell'istanza oggetto "cibo" da mangiare. Se l'oggetto è in possesso del personaggio e si tratta effettivamente di un oggetto di classe "cibo", l'oggetto è eliminato e le `ferite` del personaggio sono ridotte di una quantità corrispondente al `recupero_PF` fornito dal cibo. Ritorna void.

3.1.7 `cerca_segreti(id_pers INTEGER) RETURNS void`

Permette di rivelare connessioni tra stanze o oggetti nascosti spendendo 1 Punto Ferita. Accetta come parametro `id_pers` e ritorna void. Le `ferite` del personaggio sono aumentate di 1 e si effettua un tiro di dado virtuale 1d20 confrontato con il valore di percezione `_per` del personaggio. Se il tiro ha successo e il numero di oggetti e di passaggi nascosti è superiore a 0, allora si seleziona casualmente qualcosa da rivelare e la si rivela, rispettivamente settando `nascosto` a false o `visibile` a true. Eventuali nemici attaccano, viene quindi chiamata la funzione `attacco_nemici`.

3.1.8 `raccogli_oggetto(id_pers INTEGER, id_ogg INTEGER) RETURNS void`

Permette a un personaggio di raccogliere un oggetto, controllando che l'oggetto si trovi nella stessa stanza del personaggio e che non sia nascosto, settando inoltre `nascosto` a NULL al momento del raccoglimento. Accetta come parametri `id_pers` l'id del personaggio e `id_ogg` l'id dell'oggetto da raccogliere, ritorna void.

3.1.9 `drop_oggetto(id_pers INTEGER, id_ogg INTEGER) RETURNS void`

Permette a un personaggio di far cadere a terra un oggetto, controllando che l'oggetto sia di proprietà del personaggio e settando `nascosto` a false al momento del drop. Accetta come parametri `id_pers` l'id del personaggio e `id_ogg` l'id dell'oggetto, ritorna void.

3.1.10 `rolla_attacco(_att INTEGER, _danno INTEGER, _dif INTEGER) RETURNS INTEGER`

Accetta come parametri `_att` e `_danno` il valore di attacco e danno dell'attaccante, e come `_dif` il valore di difesa del difensore. Ritorna `_danno` se il tiro di un dado 1d20 sommato a (`_att` + `_dif`) supera 12, altrimenti ritorna 0.

3.1.11 attacco_nemici(id_pers INTEGER, id_stanza INTEGER) RETURNS void

Effettua gli attacchi dei nemici contro il personaggio. Viene chiamata quando si svolgono azioni come attaccare, spostarsi, raccogliere, cercare. Non fa altro che chiamare `rolla_attacco` per ogni nemico presente nella stanza del giocatore e con il valore restituito aumentare le `ferite` di quest'ultimo. Accetta come parametro `id_pers` l'id del personaggio e come `id_stanza` l'id della stanza in cui si trova. Ritorna void.

3.1.12 attacca(id_pers INTEGER, id_nemico INTEGER) RETURNS void

Permette di attaccare un nemico. Inizialmente controlla che il nemico non si trovi in una stanza diversa dal personaggio, poi chiama `attacco_nemici` per far subire al personaggio gli attacchi dei nemici, e infine attacca il nemico: chiama `rolla_attacco` e sottrae il valore di ritorno ai `_PFrim` del nemico. Se questi scendono sotto lo 0 il nemico viene eliminato e alle `monete` e ai `prox_PE` del personaggio vengono aggiunti rispettivamente le `monete` del nemico e il valore della somma dei suoi attributi. Accetta come parametri `id_pers` l'id del personaggio e `id_nemico` l'id del nemico attaccato. Ritorna void.

3.1.13 compra(id_pers_buyer INTEGER, id_ogg INTEGER) RETURNS void

Compra un oggetto messo in vendita: controlla se l'oggetto è effettivamente in vendita (`pr_vendita` non è NULL) e se il personaggio dispone di `monete` sufficienti, poi effettua la transazione. Per garantire il corretto trasferimento dell'oggetto e dei fondi viene utilizzata una subtransazione, implementata con il comando `plpy.subtransaction()` di PL/Python. La funzione accetta come parametri `id_pers_buyer` l'id del personaggio compratore e `id_ogg` l'id dell'oggetto da comprare. Ritorna void.

3.1.14 vendi(id_pers INTEGER, id_ogg INTEGER, prezzo INTEGER) RETURNS void

Mette in vendita un oggetto dall'inventario di un personaggio: inizialmente controlla che l'oggetto sia di proprietà del personaggio, poi lo mette in vendita ponendo `pr_vendita` uguale al prezzo, settando eventualmente anche `equip` a false. Accetta come parametri `id_pers` l'id del personaggio, `id_ogg` l'id dell'oggetto, e `prezzo` il prezzo di vendita. Ritorna void.

3.1.15 zaino(id_personaggio INTEGER) RETURNS SETOF ist_oggetto_view

Funzione che a partire dalla vista `ist_oggetto_view` permette di ottenere un elenco degli oggetti visibili (non in vendita e non consumati) nello zaino di un personaggio. Accetta come parametro `id_personaggio` l'id del personaggio di cui visualizzare lo zaino, ritorna un sottoinsieme di `ist_oggetto_view` composto dagli oggetti nello zaino.

3.2 Trigger

3.2.1 `trigger_capacita_zaino BEFORE UPDATE OR INSERT ON ist_oggetto FOR EACH ROW EXECUTE PROCEDURE funzione_capacita_zaino()`

Controlla che il numero di oggetti nello zaino di un personaggio non superi ceil(_cos/2). Per farlo chiama `zaino(id_personaggio)` e usa l'operatore aggregato COUNT. Se il limite viene superato l'operazione è annullata e l'oggetto non viene raccolto.

3.2.2 `trigger_new_att_dif BEFORE INSERT ON ist_oggetto FOR EACH ROW EXECUTE PROCEDURE funzione_new_att_dif()`

Controlla che istanze di oggetti di attacco o difesa siano creati con attributo `equip` non NULL e viceversa per gli altri oggetti, e controllo che oggetti consumabili siano creati con attributo `consumato` non NULL e viceversa per gli altri oggetti. Se viene rilevato un inserimento errato si modifica il campo inserendo NULL o false, in base alla classe del tipo di oggetto.

3.2.3 `trigger_equip_att_dif BEFORE UPDATE OR INSERT ON ist_oggetto FOR EACH ROW EXECUTE PROCEDURE funzione_equip_att_dif()`

Controlla che un personaggio non possa equipaggiare contemporaneamente due oggetti di attacco o difesa. Se prova a equipaggiarne uno, setta `equip`=false a tutti gli altri oggetti del personaggio della stessa classe di quello equipaggiato.

3.2.4 `trigger_consum_wearoff AFTER UPDATE ON personaggio FOR EACH ROW EXECUTE PROCEDURE funzione_consum_wearoff()`

Se viene rilevato un cambiamento di stanza del personaggio (campo `in_stanza`), vengono cancellati tutti gli oggetti del personaggio con l'attributo `consumato`=true. Così facendo si eliminano definitivamente gli oggetti consumati dal personaggio nella stanza, che pur essendo diventuti a lui invisibili fornivano bonus.

3.2.5 `trigger_check_cambio_stanza BEFORE UPDATE ON personaggio FOR EACH ROW EXECUTE PROCEDURE funzione_check_cambio_stanza()`

Controlla che gli spostamenti del personaggio da una stanza all'altra siano permessi, ovvero verifica che sia presente un collegamento tra le stanze nella tabella `connessa` e che `visibile`=true. Se le condizioni non sono verificate annulla l'azione.

3.2.6 `trigger_attacco_cambio_stanza AFTER UPDATE ON personaggio FOR EACH ROW EXECUTE PROCEDURE funzione_attacco_cambio_stanza()`

Quando il personaggio cambia stanza viene chiamata la funzione `attacco_nemici (id_personaggio, id_stanza)` (usando come `id_stanza` l'id della stanza precedente). Il personaggio viene quindi attaccato dai nemici in una stanza se si sposta da essa prima di averli eliminati tutti.

3.2.7 `trigger_attacco_raccogli_oggetto AFTER UPDATE ON ist_oggetto FOR EACH ROW EXECUTE PROCEDURE funzione_attacco_raccogli_oggetto()`

Quando il personaggio raccoglie un oggetto viene chiamata la funzione `attacco_nemici (id_personaggio, id_stanza)`. Il personaggio viene quindi attaccato dai nemici in una stanza se raccoglie un oggetto prima di averli eliminati tutti.

3.2.8 `trigger_item_drop_morte AFTER UPDATE ON ist_oggetto FOR EACH ROW EXECUTE PROCEDURE funzione_morte()`

3.2.9 `trigger_ferite_morte AFTER UPDATE ON personaggio FOR EACH ROW EXECUTE PROCEDURE funzione_morte()`

Entrambi i trigger utilizzano la funzione `funzione_morte()` per controllare la morte del personaggio rispettivamente dopo il raccoglimento o il drop di un oggetto (posso raccogliere un oggetto con bonus PF negativo e morire, o far cadere un oggetto che con il bonus PF teneva in vita il personaggio), e dopo il cambiamento del valore di `ferite` del personaggio. Se il valore `PF_rim` dei Punti Ferita rimanenti ottenuto da `personaggio_attr_deriv` è inferiore o uguale a 0, allora viene settato l'attributo `morto`=true. Viene poi cancellato il dungeon in cui il personaggio si trovava cancellando le stanze scorrendole per id all'indietro fino a trovare una stanza finale (da non cancellare), e in avanti fino a trovare una stanza finale (da cancellare). Vengono anche cancellati da `ist_oggetto` tutti gli oggetti di proprietà del personaggio, questo perchè a differenza di quando si termina una partita con una vittoria, la morte impedisce permanentemente al personaggio di ricominciare nuove partite.

4 Web UI

L'interfaccia di gioco è implementata in `python3` utilizzando CGI (Common Gateway Interface) per generare dinamicamente pagine in XHTML5. Per la connessione con PostgreSQL viene usato il connettore `psycopg2`.

4.1 `index.py`

`index.py` è suddiviso in una parte di inizializzazione e una di generazione delle pagine.

Inizialmente si connette al database, fa il parsing dei cookie, inizializza o riprende la sessione dell'utente, dopodichè controlla che l'utente abbia i permessi per visitare la pagina.

In base all'argomento GET `page` vengono stampate pagine differenti, corrispondenti a ogni fase di gioco.

Alla fine chiude la sessione, fa il commit su database della transazione e stampa la pagina.

4.2 Classi ausiliarie

Sono state implementate diverse classi ausiliarie per modularizzare il programma: La classe `Page` ha lo scopo di facilitare la generazione del codice XHTML5 tramite chiamate di metodi. La classe `DB` è un'interfaccia con il controllo degli errori di `psycopg2`. La classe `Session` gestisce la sessione degli utenti facendo richieste alla tabella `sessione` del database.