Project Luccio
Integrazione interfaccia

Scopo e premesse:

Questo documento ha lo scopo di essere una breve guida alla realizzazione di una interfaccia grafica compatibile con il modello e la logica di gioco già sviluppata per Project Luccio.
Non verranno esposte tutte le classi nel dettaglio, ci sarà un canovaccio generale e qualche esempio, tuttavia ci si aspetta che lo sviluppatore della gui consulti il materiale su disponibile su git qualora ne avesse necessità.

La funzione updateUi:

La prima cosa che deve realizzare lo sviluppatore dell'interfaccia è la funzione ***updateUi*** che deve essere generale, di scope globale (accessibile ad ogni elemento del codice) senza parametri in ingresso e nessun valore in ritorno.

Tale funzione viene chiamata automaticamente dalla logica di gioco quando ci sono delle variazioni che devono essere mostrate dall'interfaccia, questa funzione deve quindi ridisegnare e/o modificare gli elementi grafici per riflettere questi cambiamenti.

Per ottenere lo stato corrente la funzione ha a disposizione l'oggetto globale ***game*** il quale contiene i campi fondamentali:

- ***player*** di tipo ***Character***, contenente le informazioni sul giocatore;
- ***currentFloor*** di tipo ***Floor***, contente le informazioni sul piano (stanza) corrente.

Vediamo nel dettaglio questi oggetti, evidenzierò quelli a mio avviso più rilevanti per lo sviluppo dell'interfaccia:

player è un'istanza del tipo ***Character***, il quale contiene i seguenti membri:

- ***name*** di tipo ***String***, contiene il nome del personaggio;
- ***image*** di tipo ***String*** contiene l'url dell'immagine del personaggio (se si deciderà di usarla);
- ***bag*** di tipo ***BackPackItem*** contente l'eventuale oggetto nello zaino (più aventi verrà descritto il tipo BackPackItem);
- ***health*** di tipo ***int*** rappresentante il valore di salute da 0 a 4;
- ***hunger*** di tipo int rappresentante il valore di fame da 0 a 4 (inverso della sazietà);
- ***mood*** di tipo ***int*** rappresentante il valore di psiche da 0 a 4;
- ***enemy*** di dipo ***Character*** è un puntatore all'avversario corrente.

Consultando questi campi risulta facile disegnare l'interfaccia, per fare un breve esempio, nel caso si debba rappresentare la salute con quattro quadrati, basterà disegnare tanti quadrati fino al valore del membro health.

Mixink di Marco Botter
Via Enzo Venturini, 82
31057 - Lanzago di Silea (TV)

C.F. BTTMRC87P25L407G
P.Iva 04800470264
REA 378948

t (+39) 377 5404867
e info@mixink.it
w mixink.it

Esempio:

```
function updateUi()
{
    /*
     * Disegno di altri elementi grafici…
     */
    var i;
    for(i=0;i<game.player.health;i++)
    {
       //Disegna un quadrato…
    }
    /*
     * Altri elementi grafici…
     */
}
```

Ovviamente, la tecnica qui sopra proposta è solo un esempio, si è liberi di usare la metodologia che si preferisce o è più utile allo scopo

Con i valori numerici sopra descritti sarà molto semplice disegnare lo stato del personaggio.
Tali campi valgono anche per l'avversario, il quale è sempre un'istanza della classe Character e può essere raggiunto tramite ***game.player.enemy*** o ***game.currentFloor.enemy***, ad esempio per leggere la vita dell'avversario si andrà a consultare il membro ***game.player.enemy.health***.

L'oggetto presente nello zaino del giocatore, e quello nello zaino dell'avversario sono tutte istanze della classe ***BackPackItem***, vediamo nel dettaglio quali membri sono contenuti:

- ***name*** di tipo ***String*** è il nome dell'oggetto;
- ***description*** di tipo ***String*** è una descrizione più lunga dell'oggetto (da decidere se/dove usare);
- ***abundance*** di tipo ***int*** rappresenta la facilità di trovarlo disponibile in una stanza;
- ***type*** di tipo ***String*** è la tipologia di oggetto;
- ***value*** di tipo ***int*** è il valore di effetto base dell'oggetto;
- ***maximumUses*** di tipo ***int*** è il numero massimo di utilizzi;
- ***perform*** di tipo ***function*** è una funzione per effetti aggiuntivi (ndr: la chitarra nella demo testuale);
- ***icon*** di tipo ***String*** contenente l'url dell'icona.

Alcuni campi come ***type*** possono essere ignorati perché l'unica utilità può essere quella di verificare se il giocatore ha in mano un'arma per visualizzare o meno un pulsante di attacco, ma per questo c'è un metodo più semplice che descriverò più avanti.
Vediamo un esempio per visualizzare il nome dell'oggetto

Ipotizziamo di volere scrivere il nome dell'oggetto nello zaino del giocatore in un paragrafo con id "playerBag" e quello dell'avversario nel paragrafo "enemyBag"

Esempio:

```
function updateUi()
{
    /*
     * Disegno di altri elementi grafici…
     */
    var playerBag=document.getElementById(“playerBag”); //Ottieni paragrafo giocatore
    var enemyBag=document.getElementById(“enemyBag”);   //Ottieni paragrafo avversario
    if(game.player.bag!==null)                          //Se lo zaino non è vuoto
    {
        playerBag.innerText=”Zaino: ”+game.player.bar.name;    //Visualizza nome oggetto
    }
    else
    {
        playerBag.innerText=”Zaino vuoto”               //Scrivi che lo zaino è vuoto
    }
    if(game.player.enemy.bag!==null)                    //Se lo zaino del nemico non è vuoto
    {
        enemyBag.innerText=”Zaino: ”+game.player.enemy.bar.name;  /Nome oggetto nemico
    }
    else
    {
        enemyBag.innerText=”Zaino vuoto”                //Scrivi che lo zaino nemico è vuoto
    }

}
```

Raccomando di controllare sempre la presenza dell’oggetto tramite ***!==null*** dato che, in Javscript, il tentativo di accesso a membri di un oggetto inesistente (null pointer) causa un errore grave che comporta il crash del gioco.

Vediamo ora gli elementi di interesse del piano/stanza di gioco, contenuti nell’oggetto ***game.currentFloor***:

- ***number*** di tipo ***int*** rappresenta il numero del piano;
- ***objects*** di tipo ***array di BackPackItem*** contiene gli oggetti presenti nel piano;
- ***player*** di tipo ***Character*** è un puntatore al giocatore;
- ***enemy*** di tipo ***Character*** è un puntatore all’avversario;
- ***turn*** di tipo ***String*** informa di chi è il turno corrente;
- ***actions*** di tipo ***int*** contiene il numero complessivo di azioni rimaste;
- ***lastEnemyChoice*** di tipo ***String*** informa sull’ultima scelta fatta dal nemico;
- ***exchangeRefused*** di tipo ***int*** contiene il numero di scambi rifiutati (uso interno IA);
- ***starting*** di tipo ***bool*** risulta vero all’inzio del piano, falso poi.

Il membro turn risulta utile per capire se è il turno del giocatore e visualizzare o meno i pulsanti a lui dedicati.
Sono disponibili due costanti con i valori che può essumere: ***TURN_PLAYER*** e ***TURN_CPU***.

Vediamo un esempio di quest’uso:

Esempio:
```
function updateUi()
{
    /*
     * Disegno di altri elementi grafici…
     */
    if(game.currentFloor.turn===TURN_PLAYER)    //Se è il turno del giocatore
    {
        /*
         * Disegna pulsanti per giocatore
         */
    }
}
```

Il campo ***lastEnemyChoice*** informa sull'ultima scelta effettuata dell'avversario nel suo turno, può essere consultata per informare il giocatore o per verificare se viene proposto uno scambio.
Anche per questo membro sono disponibili delle costanti:

- ***CHOICE_IMMEDIATE_USE*** significa che l'IA ha usato il suo oggetto;
- ***CHOICE_COLLECTION*** significa che l'IA ha raccolto un oggetto;
- ***CHOICE_EXCHANGE*** significa che l'IA ti propone uno scambio;
- ***CHOICE_ATTACK*** significa che l'IA ti ha attaccato (la battaglia automatica c'è già stata);
- ***CHOICE_NONE*** significa che l'IA ti ha passato il turno.

Vediamo un esempio, se il paragrafo "info" contiene un'informativa per il giocatore:

Esempio:
```
function updateUi()
{
    /*
     * Disegno di altri elementi grafici…
     */
    if(game.currentFloor.lastEnemyChoice===CHOICE_EXCHANGE)
    {
     info.innerText="L'avversario ti propone uno scambio";   //Notifica la proposta
     /*
      * Disegna pulsanti "si" e "no"?
      */
    }
    if(game.currentFloor.lastEnemyChoice===CHOICE_ATTACK)
    {
        info.innerText="L'avversario ti ha attaccato";          //Notifica l'avvenuto attacco
    }
    if(game.currentFloor.lastEnemyChoice===CHOICE_COLLECTION)
    {
        info.innerText="L'avversario ha raccolto un oggetto"    //Notifica la raccolta
    }
    if(game.currentFloor.lastEnemyChoice===CHOICE_IMMEDIATE_USE)
    {
        info.innerText="L'avversario ha usato il suo oggetto"   //Notifica l'uso
    }
}
```

Il membro ***starting*** può essere utilizzato per verificare se ci si trova appena all'inizio del piano per visualizzare qualcosa, ad esempio, l'animazione di apertura degli occhi.

Il membro ***objects*** è un semplice array di ***BackPackItem***, del tutto analoghi a quelli che possono essere presenti negli zaini dei personaggi.
Questi sono gli oggetti che possono essere raccolti dal giocatore o dall'avversario.

Prima di passare alla programmazione delle azioni, vediamo alcuni metodi informativi accessori disponibili per i personaggi (classe Character):

- ***isDead()*** ritorna un booleano che indica se il personaggio è morto, può essere utilizzato per determinare la condizione di game over;
- ***isArmed()*** ritorna verso se il personaggio ha in mano (zaino) un'arma e quindi può attaccare.

Azioni:

Innanzi tutto, la logica di gioco avanza e gestisce tutti gli eventi automatici tramite chiamate al metodo ***game.play()***, sia nel turno del giocatore, sia in quello dell'avversario.
Questo è stato fatto per avere la possibilità e il tempo di vedere sullo stato e conseguenze delle proprie scelte e di quelle dell'avversario,
Ad esempio, a questo scopo, nella demo testuale è stato inserito il pulsante "Avanti" che chiama solo ***game.play()***.

Uso proprio oggetto:

Ovviamente verificando prima la presenza di un oggetto nello zaino, visualizzando quindi la possibilità di visualizzarlo o meno, il codice è il seguente:

Esempio:
```
game.player.use();     //Uso dell'oggetto
game.play();           //Avanza
```

Attacco:

Nella pratica l'attacco è semplicemente l'uso della propria arma, quindi si utilizza lo stesso codice dell'uso di uno oggetto

Raccolta:

Poste tutte le opportune verifiche sugli oggetti del piano presenti (o non presenti se qualcuno è stato già raccolto e/o consumato), la raccolta si effettua chiamando il metodo ***pick*** del piano, passando come parametro l'indice in base zero dell'oggetto da raccogliere (0 per il primoi, 1 per il secondo).

Vediamo un esempio, per la raccolta del primo oggetto:

Esempio:
```
if(game.currentFloor.objects[0]!==null)   //Verifica presenza oggetto
{
   game.currentFloor.pick(0);             //Raccogli oggetto
}
game.play();                              //Avanza
```

Scambio:

Lo scambio è leggermente più complesso, perché è necessario verificare se l'avversario lo accetta o meno (se l'avversario è morto accetta sempre), per fare questo si chiama il metodo dell'IA ***acceptOrNot***, e il metodo ***exchange*** per finalizzare lo scambio.
acceptOrNot ritorna un booleano, vero se l'avversario accetta, falso altrimenti,e riceve in ingresso il riferimento all'avversario e all'oggetto mentre a **exchange** vanno passati i riferimenti ai personaggi, vediamo un esempio:

Esempio:
```
if(acceptOrNot(game.currentFloor.enemy, game.player.bag)   //Se l'avversario accetta
{
   /*
    * Notifica al giocatore che lo scambio è accettato
    */
   exchange(game.player, game.currentFloor.enemy);         //Scambio
}
else
{
   /*
    * Notifica al giocatore che lo scambio è stato rifiutato
    */
}
game.play();                                               //Fai avanzare il gioco
```

Scambio proposto:

Uno scambi proposto dall'avversario si può verificare, come descritto precedentemente, controllando se il membro ***lastEnemyChoice*** del piano corrisponde alla costante ***CHOICE_EXCHANGE***
In tal caso si possono visualizzare dei pulsanti "sì/no" o qualche altra tecnica di interazione per accettare o meno lo scambio.
Per accettare si userà il seguente codice

Esempio:
```
exchange(game.player, game.currentFloor.enemy);   //Scambio
game.play();                                      //Avanza
```

Per rifiutare invece:

Esempio:
```
game.currentFloor.refuseExchange();               //Rifuta
game.play();                                      //Avanza
```

File necessari:

La pagina html dell'interfaccia dovrà richiamare nella parte ***head*** con le direttive ***script src*** i seguenti file di codice:

- utils.js
- backpackItems.js
- characters.js
- floors.js
- game.js
- intelligence.js

Al termine della parte body andrà invece inserito il codice di avvio del gioco, ovvero

Esempio:

```
<!-- Tutto il resto della pagina… -->
        <script>
           Game=new Game();
           updateUi();
        </script>
    </body>
</html>
```

Conclusioni:

La guida e gli esempi riportati dovrebbero coprire tutti gli aspetti salienti per la costruzione di un'interfaccia compatibile, alle tecniche per svilupparla al meglio come eventuali clousure, cicli diversi o altro si lascia la libera scelta allo sviluppatore della gui.
Si invita in caso di dubbi a consultare la demo testuale disponibile su git.

Mixink di Marco Botter
Via Enzo Venturini, 82
31057 - Lanzago di Silea (TV)

C.F. BTTMRC87P25L407G
P.Iva 04800470264
REA 378948

t (+39) 377 5404867
e info@mixink.it
w mixink.it