EPS-UAM

Documentación Técnica

PirateShip

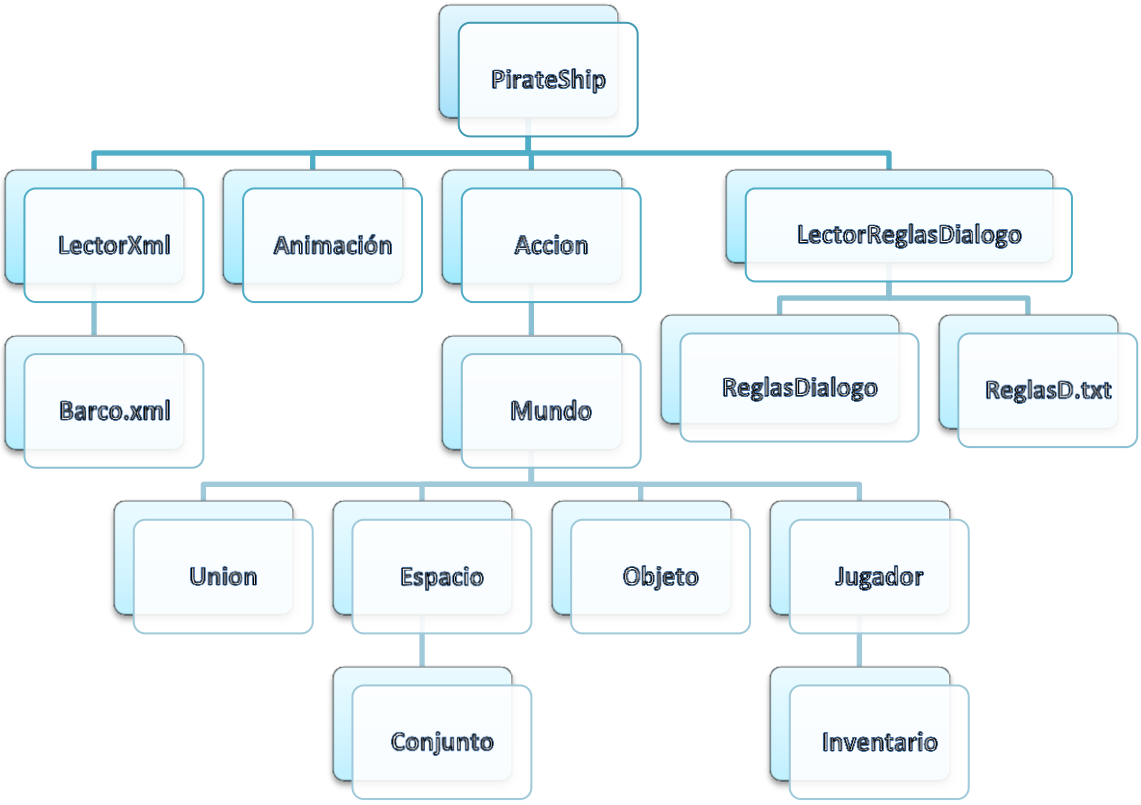
Juan Manuel Cornet, Alejandro Ortiz e Irene Colmenar

5/1/2012

PPROG-G2111

Tabla de Contenidos

Esquema del proyecto



Nota: Cada uno de los módulos dependen de Const.h

Módulo Acción

Este módulo contiene todas las funciones necesarias para trabajar con las diferentes acciones del mundo.

El TAD acción es el siguiente:

```
struct _accion {
      TipoAccion id;
      char * od;
};

enum _TipoAccion{
      JA_IR,
      JA_COGER,
      JA_DEJAR,
      JA_EXAMINAR,
      JA_ENCIENDE,
      JA_APAGA,
      JA_ABRIR,
      JA_DIALOGAR,
      JA_FINAL_BOSS
};
```

Las funciones que incluye son las siguientes:

1. `BOOL introduce_accion(char * quote, Accion * ac);`

Función que Interpreta una frase introducida por el usuario en una acción a ejecutar por el juego. En el caso en el que se produzca algún error devuelve FALSE.

2. `Accion * crea_accion();`

Función que reserva memoria para una acción y la crea.

3. `void destruye_accion(Accion *a);`

Función que elimina una acción, liberando la memoria reservada para ella.

4. `int ejecuta_accion(Mundo *m, Accion *a);`

Función que ejecuta una acción que se le pasa por argumento después de haber sido interpretada por la función introduce_accion. Devuelve un entero que indica que se debe hacer a continuación. La idea principal es que ejecuta_accion modifique el tema de conversación del main cuando el comando DIALOGAR tiene éxito. Se fue extendiendo según se necesito mas 'comunicación' entre el modulo acción y el programa principal.

Los números positivos (incluyendo 0) indican temas de conversación que el programa main debe guardar. -1 y -2 serian fracaso y éxito respectivamente a la hora de ejecutar la acción dada. -3 indica que ha habido un cambio de área con el comando IR, y por tanto el tema de conversación debe resetearse a -1 en el programa principal. Por ultimo -4 y -5 indican final de juego, siendo -4 victoria y -5 derrota.

5. `BOOL ejecuta_examina(Mundo *m, Accion *a, char *examinado, Tam max_len);`

Función que devuelve la descripción alojada en el campo examinado. Esta acción puede modificar valores como mostrar un objeto oculto.

Módulo mundo

Este módulo contiene el TAD mundo y todas las funciones necesarias para interactuar con él.

El TAD mundo tiene la siguiente definición:

```
Typedef struct _mundo {
      Espacio * espacios[MAX_ESPACIOS + 1];
      Objeto * objetos[MAX_OBJETOS + 1];
      Union * uniones[MAX_UNIONES + 1];
      Jugador *jugador;
      int vida_malo;
      int vida_prota;
      int atk_malo;
}Mundo;
```

En este TAD hemos añadido los campos vida_malo, vida_prota y atk_malo. Estos campos solo son útiles en la batalla final, y se inicializan al escoger la dificultad al iniciar el juego.

Las funciones de este módulo son las mas complejas junto con las de acción, y desde luego son las mas extensas con diferencia. Juntan gran cantidad de módulos y manejan toda la información del mundo, que es la mayor parte del juego.

Funciones de creación y destrucción del TAD:

```
Mundo * nuevo_mundo();

void destruye_mundo(Mundo * m);
```

Funciones de inicialización del TAD (para lectorXML principalmente):

```
Espacio * annade_espacio_mundo(Mundo * m, Id id);

Objeto * annade_objeto_mundo(Mundo * m, Id id);

Jugador * annade_jugador_mundo(Mundo * m);

Union * annade_union_mundo(Mundo * m, Id id);
```

Funciones de obtención de datos del mundo:

```
Espacio* dame_espacio_mundo(Mundo * m, Id id);

Objeto * dame_objeto_por_nombre (Mundo *m,const char
*nombre);

Objeto * dame_objeto_mundo(Mundo * m, Id id);

Jugador* dame_jugador_mundo(Mundo *m);

Union*   dame_union_mundo(Mundo *m,Id id);

Union * dame_union_por_nombre(Mundo *m, char *nombre);

Id dame_loc_jugador_mundo(Mundo * m);
```

```
Espacio * dame_espacio_norte(Mundo * m, Id espacio);

Espacio * dame_espacio_sur(Mundo * m, Id espacio);

Espacio * dame_espacio_este(Mundo * m, Id espacio);

Espacio * dame_espacio_oeste(Mundo * m, Id espacio);

Espacio * dame_espacio_arriba(Mundo * m, Id espacio);

Espacio * dame_espacio_abajo(Mundo * m, Id espacio);

Objeto * dame_objeto_espacio(Mundo * m, const char * nombre, Id espacio);

Id dame_loc_obj(Mundo *m, Id id);
```

Funciones de uso del mundo. Estas funciones no solo modifican el mundo o cogen datos de él, sino que lo hacen cuando se cumplen las condiciones necesarias:

```
BOOL desc_real_espacio(Mundo * m, char * desc);
```

Esta función muestra la descripción del espacio del jugador junto con los objetos que contiene (no ocultos) y la muestra por pantalla. Se llama únicamente al moverse de un espacio a otro, asique no es necesario comprobar posición ni nada.

```
BOOL mueve_personaje (Mundo * m, Direccion dir);
```

Esta función al jugador en una dirección. Se usa en el módulo acción correspondiendo con el comando IR. Comprobara que hay un espacio y que esta abierto con la función dame_espacio_norte (donde norte es la dirección que corresponda). En caso de que el espacio al que se accede sea el del malo final, se pide confirmación del movimiento.

```
BOOL examina_espacio(Mundo * m, Espacio * esp, char * desc);
```

Esta función devuelve la descripción de examinar del espacio. Se usa en el módulo acción al usar el comando EXAMINAR sin objeto directo. Puesto que solo devuelve la descripción, no contiene ninguna condición.

```
BOOL examina_objeto(Mundo * m, Objeto * obj, char * desc);
```

Esta función devuelve la descripción de examinar del objeto y muestra el objeto (deja de estar oculto). Se usa en el módulo acción al usar el comando EXAMINAR seguido de un objeto directo que no sea “INVENTARIO”. Comprueba que el objeto se encuentra o bien en el inventario o bien en el espacio en que se encuentra el jugador. También comprueba si el espacio esta iluminado y si el objeto esta oculto.

```
BOOL coge_objeto(Mundo * m, Objeto * obj);
```

Esta función coge un objeto del espacio en que se encuentre y lo coloca en el inventario del jugador. Se usa en el módulo acción al usar el comando COGER seguido del nombre del objeto. Comprueba que el objeto se encuentra en el espacio, que es móvil y que no esta oculto.

```
BOOL deja_objeto(Mundo * m, Objeto * obj);
```

Esta función coge un objeto del inventario y lo deposita en el espacio en que se encuentre el jugador. Se usa en el módulo acción al usar el comando DEJAR seguido del nombre del objeto. Comprueba que el objeto se encuentra en el inventario.

```
BOOL enciende (Mundo * m, Objeto * obj);
```

Esta función enciende un objeto. Se usa en el módulo acción al usar el comando ENCENDER seguido del nombre del objeto. Comprueba que el objeto se encuentra en el inventario o en el espacio del jugador. En caso de que así sea, se comprueba si el objeto es iluminable.

```
BOOL abre_union_mundo (Mundo *m, Union *u);
```

Esta función abre una unión. Se usa en el módulo acción al usar el comando ABRIR seguido del nombre de la unión. Comprueba que la unión limita con el espacio del jugador, que es abrible, que esta abierta y que el objeto llave se encuentra en el inventario.

```
BOOL apaga (Mundo * m, Objeto * obj);
```

Esta función apaga un objeto. Se usa en el módulo acción al usar el comando APAGAR seguido del nombre del objeto. Comprueba que el objeto se encuentra en el espacio en que se encuentra el jugador o en el inventario y que es iluminable.

```
BOOL ilumina_espacio(Mundo * m, Espacio * esp) ;
```

Esta función apaga un espacio. Se usa desde la función apaga, en el módulo mundo. Se encuentra en el módulo mundo porque debe comprobar que hay al menos un objeto encendido en el espacio o en el inventario del jugador.

```
BOOL apaga_espacio (Mundo * m, Espacio * esp);
```

Esta función enciende un espacio. Se usa desde la función enciende, en el módulo mundo. Se encuentra en el módulo mundo porque debe comprobar que no hay ningún objeto encendido en la sala ni en el inventario antes de apagar el espacio.

```
int dame_tema(Mundo *m, char *nombre);
```

Esta función devuelve el tema de dialogo de un objeto. Se usa desde el módulo dialogo con el comando DIALOGAR seguido del nombre del objeto. Comprueba que el objeto se encuentra en el inventario del jugador o en el espacio en que se encuentra.

```
BOOL examina_inventario (Mundo *m, char *cad);
```

Esta función devuelve la lista de objetos de un inventario. Se usa desde el módulo acción con el comando EXAMINAR INVENTARIO. Se encuentra en este módulo por que debe relacionar los ids del inventario con los nombres de los objetos.

```
int dame_defensa_jugador (Mundo *m);
```

Esta función devuelve la defensa del jugador. Se usa desde el módulo mundo en la función ejecuta_final.

```
int dame_ataque_jugador (Mundo *m);
```

Esta función devuelve el ataque del jugador. Se usa desde el módulo mundo en la función ejecuta_final.

```
void inicializa_dificultad (Mundo *m, char *dificultad);
```

Esta función inicializa la vida del jugador, la vida del malo y el ataque del malo. Se llama desde el módulo principal (función main) antes siquiera de mostrar el titulo inicial.

Batalla final:

```
int ejecuta_final(Mundo *m, char *cadena);
```

Los resultados de la batalla final dependen de que parte del capitán atacas, tu equipamiento (objetos en tu inventario) y de un grado de probabilidad aleatorio. Dependiendo del resultado la función devuelve un resultado u otro para que el módulo acción (quien llama a la función) muestre un mensaje u otro.

En caso de que el personaje no se encuentre en el espacio de la batalla final, se considerara que el comando es erróneo y se devolverá -1.

Las partes que se pueden atacar 'oficialmente' son CABEZA, TORSO, PIERNAS y BRAZOS. Cada una de estas partes tiene unas posibilidades de crítico, normal o parada distintas. Si no se ataca ninguno se esos sitios se considera parada directa.

En caso de ataque normal, se restara al malo una vida proporcional al ataque del jugador y se devolverá 2.

En caso de ataque critico, se restara 3 veces el daño de un ataque normal y se devolverá 1.

En caso de parada, el jugador recibirá un daño proporcional al ataque del malo menos la defensa del jugador y se devolverá 3.

Si en algún caso la vida del malo baja por debajo de 0, se devuelve 4 para indicar que el jugador ha ganado.

Si la vida del protagonista baja por debajo de 0, se devuelve 5 para indicar que el jugador ha perdido.

Módulo unión

Este módulo contiene el TAD unión y todas las funciones necesarias para interactuar con él.

El TAD unión tiene la siguiente definición:

```
Typedef struct _Union {
      char nombre[TAMANO_PAL];
      Id id;
      Id espacio1;
      Id espacio2;
      BOOL abrible;
      BOOL cerrada;
      Id key;
}Union;
```

Además de los campos que venían por defecto, hemos añadido el campo nombre y el campo key.

El campo nombre sirve para indicar que unión se abre con el comando abrir.

El campo key sirve para indicar que objeto se necesita tener en el inventario para abrir una unión.

En este módulo hay muchas funciones.

Las de crear y destruir una unión:

```
Union * nueva_union(Id id);

void destruye_union(Union * u);
```

Las de asignar valores a los campos de la unión:

```
BOOL une_espacios(Union * u, Id e1, Id e2);

BOOL establece_abrible(Union * u, BOOL b);

BOOL abre_union(Union * u);

BOOL cierra_union(Union * u);

BOOL escribe_nombreunion(Union *u, char *nombre);

BOOL escribe_key (Union * u, Id key);
```

Las de obtener valores de los campos del objeto:

```
Id devuelve_id_union(Union *u);

Id dame_espacio_unido(Union * u, Id e1);

BOOL es_union_abrible(Union * u);

BOOL es_union_cerrada(Union * u);
```

```
char* devuelve_nombreunion(Union *u);

Id devuelve_key (Union *u);
```

Y por último la que imprime los datos del objeto:

```
void imprime_union(Union * u);
```

Módulo objeto

Este módulo contiene el TAD objeto y todas las funciones necesarias para interactuar con él.

El TAD objeto tiene la siguiente definición:

```
Typedef struct _objeto {
      Id id;
      char nombre[TAMANO_PAL + 1];
      char descripcion[TAMANO_PAL + 1];
      char descMovido[TAMANO_PAL + 1];
      char descExaminar[TAMANO_PAL + 1];
      BOOL movil;
      BOOL movido;
      BOOL ilumina;
      BOOL encendido;
      BOOL oculto;
      int tema;
      int defensa;
      int ataque;
}Objeto;
```

Además de los campos que venían por defecto, hemos añadido el campo tema, el campo defensa y el campo ataque.

El campo tema indica el tema de conversación que corresponde al objeto. Al hacer objetos "parlantes" podemos ocultar conversaciones, o poder hablar con algo en cualquier momento si lo llevamos en el inventario.

Los campos defensa y ataque influyen únicamente a la hora de enfrentar enemigos (solo el malvado capitán en nuestro juego).

En este módulo hay muchas funciones.

Las de crear y destruir un objeto:

```
Objeto * nuevo_obj(Id id);

void destruye_objeto(Objeto * obj);
```

Las de asignar valores a los campos del objeto:

```
BOOL nuevo_tema_objeto(Objeto * obj, int tema);

BOOL nuevo_nombre_obj(Objeto * o, char * s);

BOOL nueva_desc_obj(Objeto * o, char * s);

BOOL nueva_desc_examinar_obj(Objeto * o, char * s);

BOOL nueva_desc_movil_obj(Objeto * o, char * s);
```

```
BOOL actualiza_movil_obj(Objeto * o, BOOL b);
BOOL actualiza_movido_objeto(Objeto * obj, BOOL movido);
BOOL actualiza_iluminable_obj(Objeto * o, BOOL b);
BOOL actualiza_ilumina_obj(Objeto * o, BOOL b);
BOOL actualiza_oculto_objeto(Objeto * obj, BOOL oculto);
BOOL establece_defensa(Objeto *o, int def);
BOOL establece_ataque(Objeto *o, int atk);
```

Las de obtener valores de los campos del objeto:

```
Id dame_id_objeto(Objeto * objeto);
int dame_tema_objeto(Objeto * objeto);
const char* nombre_objeto(Objeto *obj);
const char * descripcion_objeto(Objeto * obj);
const char * descripcion_objeto_movido(Objeto * obj);
const char * descripcion_objeto_examinar(Objeto * obj);
BOOL es_movil_objeto(Objeto * obj);
BOOL es_movido_objeto(Objeto * obj);
BOOL ilumina_objeto(Objeto * obj);
BOOL es_encendido_objeto(Objeto * obj);
BOOL oculta_obj(Objeto * o);
int dame_def_obj(Objeto *o);
int dame_atk_obj(Objeto *o);
```

Y por último la que imprime los datos del objeto:

```
void imprime_objeto(Objeto * obj);
```

Módulo Espacio

Este módulo contiene el TAD espacio y todas las funciones necesarias para interactuar con él.

El TAD espacio tiene la siguiente definición.

```
struct _Espacio {
      Id id;
      char descripcion[TAMANO_PAL + 1];
      char descExaminar[TAMANO_PAL + 1];
      Id norte;
      Id sur;
      Id este;
      Id oeste;
      Id arriba;
      Id abajo;
      BOOL luz;
      Conjunto *objetos_espacio;
};
```

Las funciones que incluye son las siguientes:

Funciones para crear y destruir un espacio:

```
1. Espacio * nuevo_espacio(Id id);
2. void destruye_espacio(Espacio * es);
```

Función para añadir una nueva descripción a un espacio:

```
3. BOOL nueva_desc_espacio(Espacio * e, char * s);
```

Función para crear una nueva unión entre espacios:

```
4. BOOL nueva_union_norte_espacio(Espacio * e, Id id);
5. BOOL nueva_union_sur_espacio(Espacio * e, Id id);
6. BOOL nueva_union_este_espacio(Espacio * e, Id id);
7. BOOL nueva_union_oeste_espacio(Espacio * e, Id id);
8. BOOL nueva_union_arriba_espacio(Espacio * e, Id id);
9. BOOL nueva_union_abajo_espacio(Espacio * e, Id id);
```

Funciones que añaden y quitan objetos de un espacio:

```
10.     BOOL annade_objeto_espacio(Espacio * e, Id id);
11.     BOOL quita_objeto_espacio(Espacio * es, Id id);
```

Función para actualizar la luminosidad de un espacio:

```
12.     BOOL actualiza_iluminacion_espacio(Espacio * e, BOOL
   id);
```

```
Funciones que devuelven las descripciones de un espacio:
```

```
13.     const char * desc_espacio(Espacio * es);
```

```
14.     const char * descexaminar_Espacio(Espacio * es);
```

Funciones que devuelven las id's de los espacios contiguos:

```
15.     Id dame_norte_espacio(Espacio * es);
16.     Id dame_sur_espacio(Espacio * es);
17.     Id dame_este_espacio(Espacio * es);
18.     Id dame_oeste_espacio(Espacio * es);
19.     Id dame_arriba_espacio(Espacio * es);
20.     Id dame_abajo_espacio(Espacio * es);
```

Función que imprime un espacio:

```
21.     BOOL imprime_espacio(Espacio * es);
```

Función que devuelve características de un espacio:

```
22.     Id devuelve_id_espacio (Espacio *es);
23.     BOOL devuelve_luz_espacio (Espacio *esp);
```

Módulo conjunto

Este módulo contiene el TAD conjunto y todas las funciones necesarias para interactuar con él.

El TAD conjunto tiene la siguiente definición:

```
Typedef struct _conjunto{
      Id elems[TAM_MAX_CNJ];
      int card;
}Conjunto;
```

Este TAD no tiene datos añadidos y cuenta con las siguientes funciones:

Las de crear y destruir un conjunto:

```
      Conjunto * nuevo_conjunto();
      void destruye_conjunto(Conjunto * cj);
```

Las de modificar los campos del conjunto:

```
      BOOL incluye_id(Conjunto * cj, Id id);
      BOOL quita_id(Conjunto * cj, Id id);
```

Las de obtener valores de los campos del conjunto:

```
      BOOL esta_id(Conjunto * cj, Id id);
      Tam tamano_conjunto(Conjunto * cj);
      Id dame_id_pos(Conjunto * cj, int posicion);
      BOOL es_vacio_conjunto(Conjunto * cj);
```

Y por último la que imprime los datos del conjunto:

```
      void imprime_conjunto(Conjunto * cj);
```

Módulo Jugador

Este módulo contiene todas las funciones necesarias para trabajar con el jugador.

El TAD acción es el siguiente:

```
struct _jugador{
      char nombre[TAMANO_PAL+1];
      Id localizacion;
      Inventario * inventario;
};
```

Las funciones que incluye son las siguientes:

1. `Jugador * nuevo_jugador();`

Función que crea un nuevo jugador reservando memoria para él. Esta función devuelve un puntero a jugador en el caso de que todo salga bien.

2. `void destruye_jugador(Jugador * j);`

Función que elimina un jugador liberando la memoria reservada anteriormente.

3. `BOOL establece_nombre_jugador(Jugador * j, char * nombre);`

Función que da nombre a un jugador mediante una cadena de caracteres introducida por uno de los parámetros.

4. `BOOL establece_maximo_obj(Jugador * j, unsigned int nobj);`

Función que establece el número máximo de objetos que puede llevar un jugador en su inventario, llamando a la función de inventario establece_maximo_objetos.

5. `BOOL establece_loc_jugador(Jugador * j, Id id);`

Función que establece la localización de un jugador mediante al id que se le pasa por uno de los parámetros.

6. `Id dame_loc_jugador(Jugador * j);`

Función que devuelve la id del jugador que se le pasa por parámetro. Si ese jugador no existe, devuelve NO_ID.

7. char * dame_nombre_jugador(Jugador * j);

Función que devuelve el nombre del jugador que se le pasa por parámetro. Si el jugador no existe, devuelve NULL.

8. `BOOL annade_objeto_jugador(Jugador * o, Id id);`

Función que añade un objeto a un jugador introduciéndolo en el inventario.

9. `BOOL buscar_obj_inventario(Jugador * j, Id id);`

Función que busca un objeto en el inventario del jugador, si lo encuentra devuelve TRUE.

```
10.     BOOL quita_obj_inventario(Jugador * j, Id id);
```

Función que saca un objeto del inventario del jugador introducido por parámetro, si no lo encuentra devuelve NULL.

```
11.     Inventario * devuelve_inventario (Jugador *j);
```

Función que devuelve el inventario del jugador introducido por parámetro.

Módulo Inventario

Este módulo contiene el TAD inventario y todas las funciones necesarias para interactuar con él.

El TAD inventario es el siguiente:

```
struct _Inventario{

      Conjunto * obj;

      int maxObjetos;

};
```

Las funciones que incluye son las siguientes:

1. `Inventario * nuevo_inventario();`

Función que crea un inventario nuevo reservando memoria para él.

2. `BOOL establece_max_objetos(Inventario*inv,int maxObjetos);`

Función que establece el número máximo de objetos que puede contener un inventario

3. `void destruye_inventario(Inventario * inv);`

Función que destruye un inventario liberando la memoria anteriormente reservada.

4. `BOOL nuevo_obj_inventario(Inventario * inv, Id id);`

Función que introduce un objeto nuevo en un inventario. Si el objeto no existiera o el inventario estuviera lleno, devuelve FALSE.

5. `BOOL buscar_obj(Inventario * inv, Id id);`

Función que busca un objeto por su id en el inventario, si no se encontrara devolvería FALSE.

6. `BOOL quita_obj(Inventario * inv, Id id);`

Función que saca un objeto del inventario, devuelve TRUE siempre que el objeto se extraiga correctamente.

Módulo lectorXML

Este módulo contiene todo lo necesario para cargar un mundo de un fichero xml. No tiene TAD y tiene pocas funciones. Su principal diferencia respecto al resto es que necesita las librerías especiales de xml para funcionar y por tanto opciones de compilación especiales también.

Las funciones de este modulo serian:

```
BOOL iniMundo(char *fichero, Mundo *m);

BOOL procesaMundo(char * nombrfich, Mundo *m);

BOOL procesaEspacio(xmlDocPtr doc, xmlNodePtr cur, Mundo *m);

BOOL procesaUnion(xmlDocPtr doc, xmlNodePtr cur, Mundo *m);

BOOL procesaObjeto(xmlDocPtr doc, xmlNodePtr cur, Mundo *m);

BOOL procesaJugador(xmlDocPtr doc, xmlNodePtr cur, Mundo *m);
```

La estructura del fichero .xml que lee este módulo es la siguiente:

```
<juego>
    <espacios>
        <espacio id=”ID”>
            <descripcion>Descripcion del espacio</descripcion>
            <edescripcion>Descripcion examinar del espacio</edescripcion>
            <norte>ID_UNION</norte>
            <sur>ID_UNION</sur>
            <este>ID_UNION</este>
            <oeste>ID_UNION</oeste>
            <arriba>ID_UNION</arriba>
            <abajo>ID_UNION</abajo>
            <objetos>
                <objeto id="ID_OBJETO"/>
```

```
                    <objeto id="ID_OBJETO"/>
                    ...
                </objetos>
                <luz valor="VERDADERO_O_FALSO"/>
            </espacio>
            ...
        </espacios>
        <uniones>
            <union id=”ID”>
                <nombre>Nombre de la union</nombre>
                <abrible valor="FALSO_O_VERDADERO"
inicial="FALSO_O_VERDADERO"/>
                <conexion_1>ID_ESPACIO</conexion_1>
                <conexion_2>ID_ESPACIO</conexion_2>
                <key>ID_OBJETO</key>
            </union>
            ...
        </uniones>
        <objetos>
            <objeto id="ID_OBJETO">
                <nombre>Nombre del objeto</nombre>
                <pdescripcion>Descripcion primer
uso</pdescripcion>
                <mdescripcion>Descripciones despues de
movido</mdescripcion>
                <edescripcion>Descripcion si se
examina</edescripcion>
                <movil valor="VERDADERO_O_FALSO" />
                <oculto valor="VERDADERO_O_FALSO" />
                <ilumina valor=" VERDADERO_O_FALSO"
inicial="FALSO_O_VERDADERO"/>
```

```
                <tema valor="Numero de tema" />
                <defensa valor="Valor de defensa" />
                <ataque valor="Valor de ataque" />
            </objeto>
            ...
        </objetos>
        <jugador>
            <localizacion>ID_ESPACIO</localizacion>
            <max_objetos>Numero maximo</max_objetos>
            <objetos>
                <objeto id="ID_OBJETO" />
                <objeto id="ID_OBJETO" />
                ...
            </objetos>
        </jugador>
</juego>
```

Módulo Lector de reglas de dialogo

Este módulo contiene una única función que lee las reglas de dialogo a partir de un fichero .txt con las funciones del modulo ReglasDialogo.

La única función de este modulo tiene el siguiente prototipo:

```
BOOL lee_reglas(char *fichero);
```

La estructura del fichero que lee este modulo es la siguiente:

Las primeras líneas del fichero no se leerán hasta que se encuentra una frase con 5 guiones y nada más. Esto permite escribir comentarios o ayudas para entenderlo.

En la linea siguiente a los 5 guiones esta el numero de reglas. Esto se usara únicamente para saber cuantas reglas se han de leer del fichero, con lo que poner un número más alto puede dar segmentation fault y poner un número más bajo conllevara que no se lean las últimas reglas.

Tras ello se escribirán todas las reglas con la siguiente estructura:

```
Linea sin leer(ayuda a identificar la regla siguiente)
Tema(-1 para las reglas sin tema)
Numero de patrones
Patrones (tantas lineas como patrones)
Numero de plantillas
Plantillas (tantas lineas como plantillas)
```

Para que las reglas que se usan para comunicar al jugador los resultados de sus distintos comandos funcionen con normalidad, las 23 primeras reglas deben corresponder con los siguientes sucesos:

```
IR_SUCCESS
IR_FAIL
COGER_SUCCESS
COGER_FAIL
DEJAR_SUCCESS
DEJAR_FAIL
EXAMINAR_SUCCESS
```

EXAMINAR_FAIL

ENCENDER_SUCCESS

ENCENDER_FAIL

APAGAR_SUCCESS

APAGAR_FAIL

ABRIR_SUCCESS

ABRIR_FAIL

DIALOGAR_SUCCESS

DIALOGAR_FAIL

ATACAR_NO (no se puede atacar)

ATACAR_CRIT

ATACAR_BASIC

ATACAR_FAIL (se recibe daño al tratar de atacar)

ATACAR_VICTORY

ATACAR_DEATH

TONTA

Modulo reglas de dialogo

Este módulo contiene el TAD Regla con todas las funciones necesarias para interactuar con él. También tiene la variable global l_reglas, que contendrá todas las reglas que se podrán usar en las conversaciones, y la variable global l_tema, que contendrá todos los temas con sus reglas respectivas.

El TAD Regla tiene la siguiente definición:

```
typedef struct _regla
{
  char ** patr_ent;
  int num_patr_ent;
  char ** plan_sal;
  int num_plan_sal;
  int ult;
}Regla;
```

Contiene un array de patrones, el número de patrones, un array de plantillas, el número de plantillas y un número que corresponde a la última plantilla utilizada.

En este módulo podemos distinguir entre las funciones de uso del TAD y las funciones que se encargan de llevar la conversación y administrar las variables globales.

Las de crear y destruir una Regla:

```
Regla *nueva_regla();

void destruye_regla(Regla *regla);
```

Las de asignar valores a los campos de la regla:

```
BOOL inicializa_patrones(Regla *regla, int patrones);

BOOL annade_patron(Regla *regla, char *patron, int pos);

BOOL inicializa_plantillas(Regla *regla, int plantillas);

BOOL annade_plantilla(Regla *regla, char *plantilla, int
pos);
```

Funciones que modifican las variables globales:

```
void inicializa_reglas();

void destruye_reglas();

BOOL annade_regla_temas(int tema, int regla);

BOOL annade_regla_reglas(Regla *regla, int posicion);
```

```
BOOL annade_regla(Regla *regla, int tema, int posicion);
```

Funciones para llevar la conversación:

```
extern char * selecciona_plan_sal(int ind_regla);

extern int busca_regla(int tema, const char *txt_ent);

extern char * selecciona_plan_azar(int ind_regla);

extern int tam_patron_regla(int ind_regla, int
ind_patr_entr);
```

Modulo animación

Este módulo contiene las animaciones iniciales y finales del juego. Únicamente constan de imprimir por pantalla montones de datos, por lo que, debido al tamaño que ocupan. Hemos decidido tenerlas en un módulo aparte.

Las tres funciones de este módulo serían:

```
void despedida();

void presentacion();

void ejecuta_victoria(char *nombre);

void ejecuta_derrota();
```

Manual de Juego

Bienvenido a "Pirate Ship", un juego de aventura en el que tendrás que ir recolectando objetos (algunos mas útiles que otros) para seguir avanzando. A continuación te presentamos las instrucciones que puedes realizar, así como algunos consejos de juego:

Primero de todo, debes tener la carpeta con todos los ficheros del juego. Una vez que la tengas, tienes que acceder a ella con una terminal de Linux (se recomienda utilizar la terminal en modo pantalla completa para disfrutar al máximo de las animaciones y de las imágenes del juego, si no, algunas se verán cortadas). Cuando estés dentro de la carpeta, si escribes el comando "make" generaras todos los archivos necesarios para jugar.

A continuación, puedes escribir "make clean" para borrar los ficheros generados en pasos intermedios y que la carpeta con los ficheros no contenga mas archivos de los necesarios. Ahora viene el juego: para ejecutarlo, escribe en tu consola el siguiente comando: "./PirateShip ReglasD.txt Barco.xml". Con esto ya tendrás el juego corriendo en tu consola de Linux, ahora, ¡lee el resto de las instrucciones para poder disfrutar del juego!

Las instrucciones para desenvolverte en el mundo son las siguientes:

-IR [DIRECCION CARDINAL/ARRIBA/ABAJO] -> Con esta instrucción puedes dirigirte a las cuatro direcciones cardinales (N, S, E, O) así como arriba y abajo.

-COGER [NOMBRE OBJETO] -> Con esta instrucción puedes coger un objeto de la estancia actual en la que te encuentras, siempre y cuando el objeto se pueda coger.

-DEJAR [NOMBRE OBJETO] -> Funciona del mismo modo que la anterior, solo que realiza la función contraria.

-ENCENDER [NOMBRE OBJETO] -> Enciende un objeto siempre y cuando este se encuentre en tu inventario o en la sala actual y pueda encenderse.

Esto te proporcionara luz en todas las salas que visites (si posees el objeto iluminado) o en la sala en la cual se encuentre el objeto.

-APAGA [NOMBRE OBJETO] -> Funciona exactamente igual que la instrucción anterior, solo que realiza la función contraria.

-EXAMINAR / EXAMINAR [NOMBRE OBJETO] -> Esta función sirve para obtener información detallada sobre un espacio (si, como en el primer caso, solo escribes examinar) o bien sobre un objeto si a continuación indicas el nombre del objeto a examinar.

-ABRIR [NOMBRE UNION] -> Esta función permite al jugador abrir una unión siempre y cuando tenga el objeto necesario para abrirla.

Una unión no es más que el comunicador entre dos estancias contiguas mediante la que se puede pasar de una a otra.

-DIALOGAR [NOMBRE PERSONAJE] -> Esta función te permite activar el dialogo con los personajes del juego. Mediante ellos puedes conseguir información útil.

Una vez conocidos los comandos, te interesará ver las siguientes sugerencias y notas para que el juego te resulte lo más sencillo posible:

-Algunos objetos se encuentran ocultos, esto quiere decir que si encuentras la sala en la que se hallan, primero deberás examinarlos para poder cogerlos.

-Siempre hay manera de encontrar donde están los objetos ocultos. Solo requiere paciencia y buscar información sobre ellos.

-Los nombres de las uniones que te interesa poder abrir te los ira dando el juego a medida que avances, ¡así que estate atento!

-Todos los objetos/uniones/personajes del juego aparecen nombrados en mayúsculas, por lo que debes mantenerte alerta para saber cuando aparecerá uno nuevo.

-Solo podrás examinar los objetos o las estancias en un sitio dotado de luz, es decir, un espacio iluminado.

-Todos los espacios se iluminan (y permanecen iluminados) una vez entres en ellos con un objeto que emita luz.

-Ten en cuenta que tu inventario tiene un limite de 10 objetos, asique si te sobrecargas deberás ir dejando los objetos que ya no te sean útiles.

-Para finalizar, una ayuda: si no tienes buena memoria, siempre puedes escribir EXAMINAR INVENTARIO para examinar los objetos que posees actualmente.

Tras haber leído este manual de juego, esperamos que la experiencia “Pirate Ship” te sea lo mas divertida y sencilla posible, ¡que te diviertas!

Ahhhh! Casi se me olvida: no te asustes si algún personaje del juego parece demasiado agresivo, ¡en el fondo todos son de ayuda!