Afirmaciones provisionales, pueden cambiar o se tienen que especificar u explicar en el glosario.

**X.BASE DE DATOS**

**X.1.TABLASenemigolla el enemigoieza la batalla el usuario.ientes y el escenariose guarda como una cadena de caracteres muy larga, para el**

La base de datos de este proyecto esta compuesta por un total de 8 tablas

Todas las tablas tienen un clave primaria compuesta por un numero

(Campos/columnas y filas es una palabra a definir en el glosario)

-**SHIP.** Esta tabla tiene los datos de las naves del juego, tiene un total de 10 campos o columnas:

-**ID\_Defense**, la clave primaria de la nave.

-**name**, nombre de la nave, formato VARCHAR, este es unico.

-**metal\_Cost**, coste de metal de construir la nave, en formato INTEGER

-**crystal\_Cost**, coste de cristal de construir la nave, en formato INTEGER, si bien los datos de cristal estan presentes, no se han implementado en el proyecto

-**deuterium\_Cost**, coste de deuterio de construir la nave, en formato INTEGER

-**initial\_Armor**, armadura inicial con la que cuenta la nave, en formato INTEGER , con este dato junto con el nivel tecnologico se calculara la armadura final

-**armor**, nivel de armadura final, en formato INTEGER (? la armadura no deberia ser necesario guardarla, pues con la tecnologia y la inicial se puede calcular la armadura final)

-**baseDamage**, el daño base que inflije la nave, en formato INTEGER , el daño final estar también influido por el nivel tenológico

-s**peed,** la velocidad, en formato INTEGER

-**generateWasting,** la probabilidad de generar residuos, en formato INTEGER con un valor de entre 0 y 100

-**DEFENSE,** esta tabla tiene los datos de las estructuras defensivas. Sus columnas de datos son exactamente las mismas que las de SHIP con las mismas propiedades

-**USER.** Esta tabla tiene el objetivo de guardar datos de los usuarios (explicar en glosario que usuario = jugador). Tiene un total de 4 campos:

-**ID\_User**. La id del usuario.

-**username**. El nombre del usuario, es un VARCHAR y tiene la propiedad de que es unico y obligatorio, por lo tanto no podran haber dos usuarios con el mismo nombre.

-**password**. La contraseña que habra registrada para el usuario, en formato VARCHAR y obligatoria.

-**birth\_date**. La fecha de nacimiento del usuario, en formato DATE, también es un campo obligatorio.

-**PLANET.** Datos de los planetas, estos pertenecen a los usuarios y almacenan recursos, naves y estructuras defensivas y tienen un nivel tecnológico, que influira en la fuerza de las naves y las estructuras. Tiene un total de 19 campos o columnas:

**-ID\_Panet**. La id del planeta.

**-User\_ID\_User**. La id del usuario al que pertenecera el planeta, es una clave foranea que liga la tabla PLANET con la tabla USER, es obligatoria y esta en formato INTEGER, como todas las IDs.

**-planet\_Name**. Nombre del planeta, en formato VARCHAR, obligatorio y también unico para que no se repitan nombres.

**-quantity\_Metal**. Cantidad de metal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-quantity\_Crystal**. Cantidad de cristal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-quantity\_Deuterium**. Cantidad de cristal que hay en este planeta, formato INTEGER, si no se especifica un valor por defecto se pone a 0.

**-num\_LighHunter**. El numero de naves de tipo LightHunter que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_HeavyHunter**. El numero de naves de tipo HeavyHunter que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_BattleShip**.El numero de naves de tipo BattleShip que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_ArmoredShip**. El numero de naves de tipo ArmoredShip que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_MissileLauncher**. El numero de esturcturas defensivas de tipo MissileLauncher que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_IonCannon**. El numero de esturcturas defensivas de tipo IonCannon que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-num\_PlasmaCannon**. El numero de esturcturas defensivas de tipo PlasmaCannon que hay en el planeta, en formato INTEGER, si no se le especifica valor por defecto se pone a 0.

**-current\_LevelDefense**. El nivel de tecnologia defensiva actual que hay en el planeta, esta en formato INTEGER y por defecto, en caso de que no se especifique un valor, es 1, el primer nivel.

**-cost\_DefenseUp**. El coste de deuterio de subir un nivel a tecnologia de defensa del planeta, es una variable en formato INTEGER y obligatoria.

**-current\_LevelAttack**. El nivel de tecnologia ofensiva actual que hay en el planeta, esta en formato INTEGER y por defecto, en caso de que no se especifique un valor, es 1, el primer nivel.

**-cost\_AttackUp**. El coste de deuterio de subir un nivel a tecnologia de ataque del planeta, es una variable en formato INTEGER y obligatoria.

(Constantes en el glosario)

-**CONSTANTS.** Tabla para guardar los valores de las constantes que regularan varios aspectos del balance del juego. Esta tabla solo tiene 3 campos:

**-ID\_Constant**. La clave primaria de la constante.

**-name**. El nombre de la constante, esta en formato VARCHAR y es unico y obligatorio.

**-value**. El valor de la constante, en formato VARCHAR y también obligatorio.

-**UNITS**. Cada nave y cada estructura es unica y por lo tanto estara registrada en esta tabla, que guardara de que tipo es la unidad y a quien pertenece. Tiene 7 campos o columnas:

**-ID\_Unit**.

**-defense\_Level**.

**-attack\_Level**.

**-Defense\_ID\_Defense**.

**-Ship\_ID\_Ship**.

**-Enemy\_ID\_Enemy**.

**-Planet\_ID\_Planet**.

-**ENEMY.** para guardar datos de los enemigos, estos son como una mezcla simplificada del jugador y el planeta, tienen naves (no disponen de estructuras defensivas), niveles tecnologicos y recursos para hacer naves, por si deseamos implementar una mecanica completa de posesión de unidades y uso de recursos como la del jugador. Esta tabla tiene un total de 11 columnas o campos:

**-ID\_Enemy**. La id del enemigo, con el mismo formato INTEGER de todas.

**-name**. El nombre del enemigo, en formato VARCHAR, obligatorio y que no puede r epetir-se.

**-quantity\_Metal**. La cantidad de metal de que dispone ese enemigo, formato INTEGER, valor 0 por defecto si no se le especifica un valor.

**-quantity\_Crystal**. La cantidad decristal de que dispone ese enemigo, formato INTEGER, valor 0 por defecto si no se le especifica un valor.

**-quantity\_Deuterium**. La cantidad de deuterio de que dispone ese enemigo, formato INTEGER, valor 0 por defecto si no se le especifica un valor.

-**num\_LightHunter**. Número total de unidades del tipo LighHunter de que dispone ese enmigo en su flota.

**-num\_HeavyHunter**. Número total de unidades del tipo HeavyHunter de que dispone ese enmigo en su flota.

**-num\_BattleShip**. Número total de unidades del tipoBattleShip de que dispone ese enmigo en su flota.

**-num\_ArmoredShip**. Número total de unidades del tipo ArmoredShip de que dispone ese enmigo en su flota.

**-current\_LevelDefense**. El nivel tecnológico defensivo que tiene este enemigo, aspecto que repercute en el poder defensivo y ofensivo de sus naves.

**-current\_LevelAttack**. El nivel tecnológico ofensivo que tiene este enemigo, aspecto que repercute en el poder defensivo y ofensivo de sus naves.

-**BATTLE**, para guardar las batallas, esta es la más compleja, se une con llaves foráneas a un usuario, un planeta y un enemigo, siendo estos tres campos requisitos para conformar los contendientes y el escenario. Guarda el reporte “step by step” o paso por paso en forma de conjunto de caracteres, y tiene dos booleanos para guardar quien gana, si el de usuario es cierto y el de enemigo es falso, gana el usuario, en caso contrario gana el enemigo. Tiene datos de residuos generados y registros del número de unidades con el que empiezan y terminan tanto jugador como enemigo.

**-ID\_Battle**. La id de la batalla.

**-User\_ID\_User**. Clave foránea que corresponde a la id del usuario que participa en la batalla, obligatoria.

**-Enemy\_ID\_Enemy**. Clave foránea que corresponde a la id del enemigo que participa en la batalla, también en formato INTEGER y obligatoria.

**-Planet\_id\_Planet**. La batalla se lleva a cabo en un planeta, este dato es otra clave foránea, la del planeta donde se lleva a cabo la batalla, formato INTEGER y obligatoria.

**-report\_StepByStep**. Aquí se guarda el reporte entero de los pasos de la batalla, este reporte se guarda como una cadena de caracteres muy larga, para ello se ha seleccionado el tipo de variable LONG. Es otro campo obligatorio.

**-user\_Winner**. Esta es una variable booleana para guardar si el usuario ha ganado o no, por problemas con la variable de tipo BOOLEAN, se ha decidido emplear una variable de tipo INTEGER que se usara de forma que 0 sea igual a false y 1 a 0.

-**enemy\_Winner**. Esta es una variable booleana para guardar si el enemigo ha ganado o no, por problemas con la variable de tipo BOOLEAN, se ha decidido emplear una variable de tipo INTEGER que se usara de forma que 0 sea igual a false y 1 a 0. Si el booleano de usuario es cierto, el de enemigo será falso y viceversa. Esta y la anterior son también campos obligatorios.

**-waste\_Metal**. Cantidad de residuos de metal que se han generado tras la batalla, en formato INTEGER y obligatorio.

**-waste\_Deuterium**. Cantidad de residuos dedeuterio que se han generado tras la batalla, en formato INTEGER y obligatorio.

**-UI\_LightHunter**. El numero inicial de unidades LightHunters con los que empieza la batalla el usuario.

**-UI\_HeavyHunter**. El numero inicial de unidades HeavyHunters con los que empieza la batalla el usuario.

**-UI\_BattleShip**. El numero inicial de unidades BattleShip con los que empieza la batalla el usuario.

**-UI\_ArmoredShip**. El numero inicial de unidades ArmoredShip con los que empieza la batalla el usuario.

**-UI\_MissileLauncher**. El numero inicial de estructuras defensivas MissileLauncher con los que empieza la batalla el usuario.

**-UI\_IonCannon**. El numero inicial de LightHunters con los que empieza la batalla el usuario.

**-UI\_PlasmaCannon**. El numero inicial de LightHunters con los que empieza la batalla el usuario.

**-UF\_LightHunter**. El número final de unidades LightHunters con los que termina la batalla el usuario.

**-UF\_HeavyHunter**. El número final de unidades HeavyHunters con los que termina la batalla el usuario.

**-UF\_BattleShip**. El número final de unidades BattleShip con los quetermina la batalla el usuario.

**-UF\_ArmoredShip**. El número final de unidades ArmoredShip con los que termina la batalla el usuario.

**-UF\_MissileLauncher**. El número final de estructuras defensivas MissileLauncher con los que termina la batalla el usuario.

**-UF\_IonCannon**. El número final de LightHunters con los que termina la batalla el usuario.

**-UF\_PlasmaCannon**. El número final de LightHunters con los que termina la batalla el usuario.

**-EI\_LightHunter**. El número inicial de unidades LightHunters con los que empieza la batalla el enemigo.

**-EI\_HeavyHunter**. El número inicial de unidades HeavyHunters con los que empieza la batalla el enemigo.

**-EI\_BattleShip**. El número inicial de unidades BattleShip con los que empieza la batalla el enemigo.

**-EI\_ArmoredShip**. El número inicial de unidades ArmoredShip con los que empieza la batalla el enemigo.

**-EF\_LightHunter**. El número final de unidades LightHunters con los que termina la batalla el enemigo.

**-EF\_HeavyHunter**. El número final de unidades HeavyHunters con los que termina la batalla el enemigo.

**-EF\_BattleShip**. El número final de unidades BattleShip con los quetermina la batalla el enemigo.

**-EF\_ArmoredShip**. El número final de unidades ArmoredShip con los que termina la batalla el enemigo.

**X.2.PROCEDIMIENTOSenemigolla el enemigoieza la batalla el usuario.ientes y el escenariose guarda como una cadena de caracteres muy larga, para el**

Para iniciar la base de datos (Iniciar BBDD en glosario) hay un conjunto de procedimientos básicos que se tienen que ejecutar. De esta forma tendremos la base de datos lista para guardar registros del juego.

**-DROP\_TABLE.** Borra todas las tablas del proyecto.

**-CREATE\_TABLE.** Crea de nuevo las tablas del proyecto.

**-INSERT\_DATA.** Inserta los datos de las naves y las estructuras defensivas en la base de datos. Estos son en total 4 filas o registros para SHIP y 3 filas para DEFENSE.

**-INITIALIZE.** Este proceso llama a los 3 procedimientos anteriores: DROP\_TABLE, para eliminar las tablas, paso previo a rehacer la base de datos en su estado inicial. Posteriormente llama a CREATE\_TABLE para crear las tablas. Por ultimo llama al procedimiento INSERT\_DATA para insertar los datos de las naves y las estructuras defensivas.

De esta forma tenemos la base de datos rehecha y con los únicos datos de naves y estructuras, que no cambiaran, por lo que podríamos en cierto modo entenderlos como constantes. El resto de datos se tendrán que ir insertando desde la ejecución del código de Java.

(Glosarios, concepto de registro o fila, como diferente de columna o campo)

Una vez creadas las tablas, hay una serie de procedimientos comunes entre ellas para hacer las operaciones de insertar registros o filas, obtener la información de un registro, modificar los datos de un registro que ya existe y por ultimo eliminar un registro que ya existe.

Estos procedimientos son 4: Los de tipo GET, los de tipo INSERT, los de tipo SET y los de tipo DELETE. La nomenclatura que siguen es el nombre del tipo de procedimiento mas el nombre de la tabla, por ejemplo para las tabla UNITS tenemos GET\_USER, INSERT\_USER SET\_USER y DELETE\_USER. Estos procedimientos actúan todos a nivel de una sola fila o registro. Se fundamentan en la ID numérica para saber sobre que fila operan, que se le especifica en los parámetros de entrada o se crea automaticamente en los procedimientos de tipo INSERT.

Las tablas de SHIP y DEFENSE solo disponen del método GET ya que su única función es servir para consultar unos datos. Las otras 6 tablas además de GET, disponen de INSERT, SET y DELETE para abrir posibilidades de trabajo y pruebas, aunque no todas las tablas necesitaran todos estos procedimientos en la ejecución del código principal una vez preparado el proyecto.

Los 4 procedimientos comunes (TABLA = Nombre de la tabla sobre la que queramos trabajar) son

-**GET\_TABLA(id IN, campo1 OUT , campo2 OUT , campo3 OUT, …)**

Este procedimiento tiene la función de a partir de una id numérica que le entramos, devolvernos todos los datos de la fila o registro de esta id. Debido a eso, la id que le entramos es una variable de tipo IN y el resto serán variables que se devuelvan, por ello son variables de tipo OUT.

La estructura básica de este procedimiento es se llama y se le especifican la id de entrada y las variables de salida, cuyo numero variara según la tabla sobre la que trabaje. Al estar todas estas variables declaradas, en la parte de declaración de variables (IS) se declaran únicamente una variable tipo numérica (id\_Encontrada) para comprobar si la id que hemos entrado existe y una excepción (excepción\_Id) hecha para activarse en caso de que la id no se haya encontrado. La variable numérica se obtiene haciendo un conteo de la cantidad de ids que hay en esta tabla, los resultados posibles son 0, no esta esta id en la tabla, o 1, se ha encontrado una sola fila con esa id. Si no se ha encontrado se activa la excepción.

Este sistema para comprobar la existencia de la id es el mismo en los otros procedimientos, salvo en los INSERT donde el requisito para continuar es que la id no exista (pues se inserta y no puede estar repetida)

Si la excepción no se ha activado, entonces la ejecución del procedimiento continuará y se hará un SELECT de todos los parámetros de la fila de datos de esa id, excepto la id misma. Estos parámetros seleccionados se depositarán en las variables de salida, de esta forma los valores estarán disponibles en la salida del procedimiento para ser recogidas por el código de Java.

**-SET\_TABLA(id IN, campo1 IN , campo2 IN , campo3 IN, …)**

Este procedimiento es para modificar un registro que ya existe. Para especificar que registro se quiere modificar, el procedimiento pide que le entremos la id, y luego que le entremos para cada columna o campo, un valor, cada uno de estos valores será el nuevo valor que actualizaremos. Si a alguno de estos campos le entramos un valor nulo (‘’) este procedimiento no actualizara este campo y mantendrá el valor antiguo, de esta forma es posible modificar solo algunos campos, excepto la id, que siempre será fija e inmutable para el registro.

Para cada variable que se actualizará se declara en el (IS) una variable con el mismo nombre y el añadido \_Insertar. Cada una de estas variables será la que se insertara. El porque de esta aparente duplicidad se explicara a continuación. Aparte de las variables de entrada y las de inserción, también declara una variable para comprobar si existe la id y una excepción por si la id no se ha encontrado. Además declara una variable de tipo VARCHAR será el script que se ejecutara posteriormente con el comando *execute immediate*.

Primero hace las comprobaciones de que la id a modificar existe, si existe procede a hacer una comporbación para cada variable que queremos actualizar. Esta comprobación consiste en ver si la variable que queremos entrar como nuevo valor en este campo o columna es nula. Si la variable de entrada no es nula, el valor que se insertara es el que hemos entrado y se hará variable\_Entrada = variable\_Entrada\_Insertar, para hacer el procedimiento de actualización quequeríamos hacer originalmente. En caso de que hayamos entrado un valor nulo, la variable que insertaremos adquirirá el valor que había previamente en la columna en ese registro.

Por ejemplo para el campo contraseña, si al procedimiento le entramos una nueva contraseña, lo actualizara ya que la variable que insertaremos para actualizar cogerá el valor de contraseña que le hemos entrado, en caso de que se entra un valor nulo para la contraseña, se insertara el valor de contraseña antiguo y a efectos prácticos no habrá habido modificación de la contraseña.

**-INSERT\_TABLA(id IN, columna1 IN , columna2 IN , columna3 IN, …)**

Procedimiento para crear un nuevo registro, incluso admitiendo especificar la ID (aunque no admitirá IDs repetidas) y especificando el resto de datos. Inserta una nueva fila. Importante diferenciar entre SET, para actualizar una fila ya existente y INSERT para crear una nueva fila.

Si no se quiere insertar un valor para todas las columnas, mirar en el PDF con el modelo relacional, los datos que tienen un asterisco rojo son datos obligatorios que no pueden estar nulos.

**-DELETE\_TABLA(id IN)**

Simple, solo nos pide una ID y a partir de esta elimina la fila.

Para la tabla constants he creado un procedimiento para obtener valores de las variables a partir del nombre, que coincidirá con el nombre de variable en Java, para hacer la búsqueda mas fácil sin necesitar de poner una Query de por medio que nos transforme el nombre en ID y de esta obtengamos el valor. Si hace falta creare mas procedimientos específicos.

Estructuras de los 4 tipos de PROCEDIMIENTO básicos (Estos son lo mínimo que tendrán, común con los otros procedimientos del mismo tipo):

-**GET\_TABLA(id IN, columna OUT , columna OUT , columna OUT, …)**

Se declaran en el procedimiento una variable por columna, todas del mismo tipo que la columna a la que correspondan.

En IS se declara una variable NUMBER(1) para comprobar si el ID existe y una excepción para disparar en caso de que el id n exista.

Hace el selecto COUNT(ID) que coincida con el ID de entrada, si no lo encuentra (id\_Encontrada =0) salta un error.

En caso contrario selecciona todas las columnas de la tabla EXCEPTO la de la ID y deposita sus valores (INTO) el resto de variables de la declaración del procedimiento, las que están en OUT.

Tiene dos excepciones, la de Id, que da el mensaje de que no se ha encontrado esta ID en la tabla y una WHEN OTHERS que incluye un mensaje especifico del error.

**-SET\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

El procedimiento mas complejo. Empieza en la declaración del procedimiento con una variable por cada columna, todas IN.

La declaración de variables IS añade para cada una de las variables de la declaración del proceso una variable idéntica, que tendrá el mismo nombre + “\_Insertar”, esta será la variable que se insertara, a continuación veremos el porque de esta variable, pero como avance, serán las que se usaran para insertar aunque parezca una redundancia.

El IS o declaración también incluye un NUMBER(1) y una excepción para el caso en que no encuentre la ID en la tabla y un VARCHAR de considerable tamaño para guardar la query que se ejecutara para actualizar la base de datos.

El procedimiento empieza con la típica comprobación de que la ID existe y el condicional con la excepción.

A continuación, tiene un condicional por cada columna de la tabla salvo la de la ID. Este condicional se activa si la variable entrada en el procedimiento es nula, en este caso lo que hace es que la variable insert adquiera el valor que tiene en la tabla esa columna en la ID que hemos entrado.

Esto hace que insertemos el valor que había previamente y por lo tanto no cambie ese dato, de esta forma si uno de los datos entrados en el procedimiento es nulo (‘’) no cambia. En caso contrario la variable a insertar será la que hemos entrado en el procedimiento.

He tenido que hacer eso y complicar el codigo porque con variables que se entran al procedimiento en IN no se les pueden asignar valores desde una query con INTO (con las variables en OUT, de salida, si que se puede hacer), pero he considerado que es útil para permitir modificar solo algunos campos de la fila y dejar otros intactos.

Terminados los condicionales, se prepara el script para actualizar, que es un UPDATE BATTLE VALUE() con valores para actualizar una fila especificando un WHERE que será donde la ID que hayamos entrado.

El script se ejecuta con execute immediate, mas seguro que escribirlo directamente. Tras todo esto tiene un mensaje de confirmación con las nuevas variables y un COMMIT.

De excepciones tiene la de ID y la WHEN OTHERS, destacar que las excepciones por precaución tienen un ROLLBACK para deshacer posibles cambios a medias como buena practica.

**-INSERT\_TABLA(id IN, columna IN , columna IN , columna IN, …)**

Para crear un nuevo registro, también nos pide que le entremos, IN, una variable por cada columna, incluida una ID, que será la ID de ese nuevo registro (Tiene codigo para comprobar que no se repita).

El IS tiene un NUMBER(1) y una excepción para comprobar que la ID no este ya insertada en la tabla y un VARCHAR del script con los valores que se insertaran.

Al empezar el procedimiento, se comprueba primero que no exista la ID, si existe id\_Encontrada =1 salta una excepción. Posteriormente prepara el script, INSERTO INTO TABLA VALUES() una fila. Executa el script con execute immediate y suelta un mensaje de confirmación que el insert esta correcto con los valores. Tiene un COMMIT y dos excepciones con ROLLBACK.

**-DELETE\_TABLA(id IN)**

El procedimiento mas sencillo de todos, solo le entramos una ID. Nos controla con la variable NUMBER(1) y la excepción que existe dicha ID y en caso de existir prepara un Script en forma de VARCHAR tipo DELETE FROM TABLA WHERE id = id\_Entrada y executa el script. Acaba en COMMIT o ROLLBACK en las dos excepciones típicas.