**Algoritmos y Estructuras de Datos**

**Laboratorio 2 Unidad 1**

Nelson López

Carlos Lizalda.

Santiago Chasqui.

**1. Identificación del problema:** La empresa Epic Games solicita el desarrollo de ciertas mejoras a su juego bandera conocido como Fortnite, las cuales deberán estar incluidas en la nueva versión del software y que serán descritas con mayor detalle a continuación:

|  |  |
| --- | --- |
| **R1** | Categorizar jugadores por su destreza de juego |
| Resumen | Crea un ranking de jugadores categorizándolo por la destreza de cada jugador |
| Entrada | Jugadores |
| Salida | Ranking de Jugadores |

|  |  |
| --- | --- |
| **R2** | Crear partidas exclusivas. |
| Resumen | Crea una partida exclusiva de acuerdo con el tipo de plataforma a la que pertenezca el usuario. |
| Entradas | Jugadores |
| Salida | Partida exclusiva de jugadores |

Falta especificar el requerimiento sobre la latencia

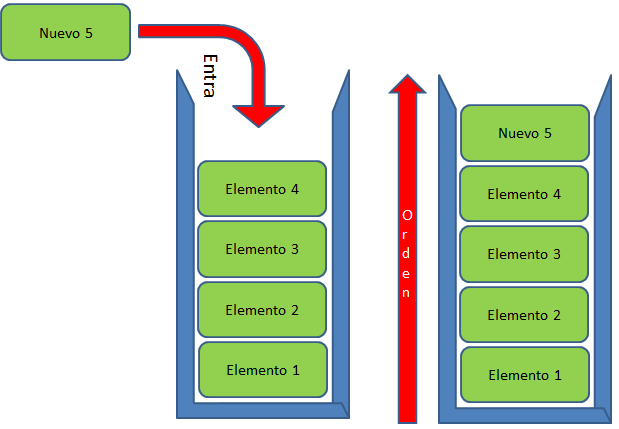
|  |  |
| --- | --- |
| **R3** | Crear partida especial |
| Resumen | Se crea una tipo de partida específica para el evento de San Valentin con una jugabilidad distinta a la utilizada comúnmente. |
| Entradas | Jugadores |
| Salidas | Partida especial. |

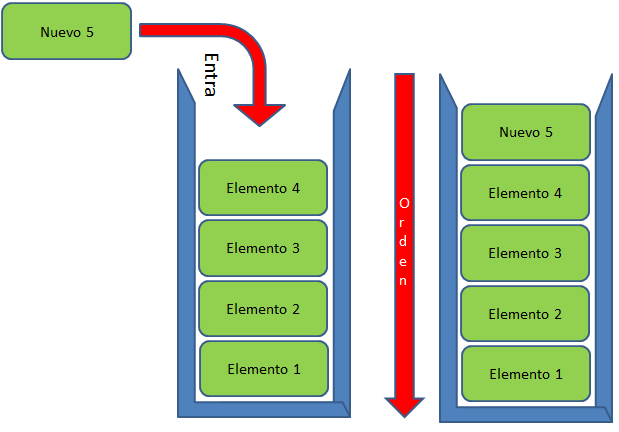
Requerimientos no funcionales:

* Los jugadores de una misma partida deben tener una latencia muy cercana.

**2. Recopilación de la información:** A continuación se describirán los conceptos formales necesarios para abordar el problema de manera pertinente:

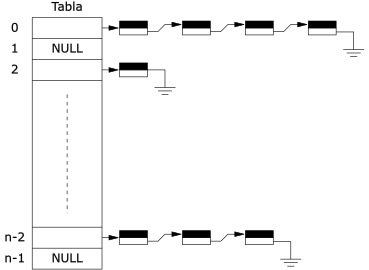
* **Estructura de Datos:** En términos sencillos, una estructura de datos una forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación. Entre las estructuras de datos mas conocidas se encuentran:
  + **Pilas:** Una pila es una colección ordenada de datos a los que solo se puede acceder por un extremo, denominado tope o cima de la pila. Esta es una estructura de datos que se caracteriza porque el último elemento en entrar es el primero que debe salir. Este tipo de estructura se denomina LIFO (Last-In-First-Out). La siguiente es una representación gráfica de esta estructura de datos:

(Blanco, 2017)  
(Blanco, 2017)

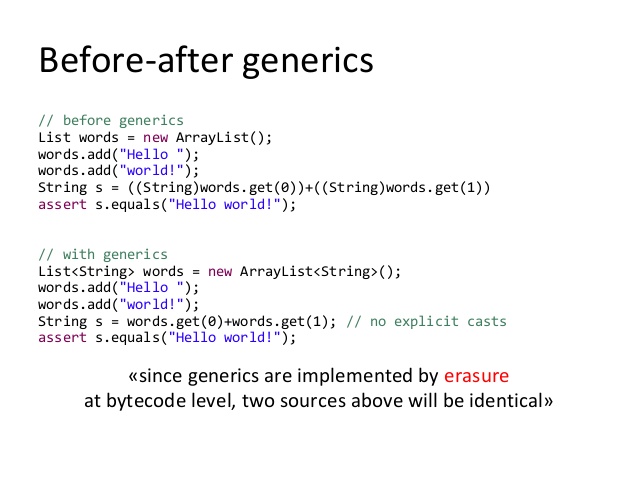
* + **Colas:** Es una estructura de datos que se caracteriza porque el primer datos en entrar es el primero en salir. Este tipo de estructuras se conocen como estructuras FIFO. La siguiente es una representación gráfica de esta estructura de datos:

(Blanco, 2017)

* + **Tablas Hash:** “Es una estructura de datos no lineal cuyo propósito final se centra en llevar a cabo las acciones básicas (inserción, eliminación y búsqueda de elementos) en el menor tiempo posible, mejorando las cotas de rendimiento respecto a un gran número de estructuras.” (Heileman, 1994). La siguiente es una representación gráfica de esta estructura de datos:

 (uc3m, 2017)

* + **Interfaz:** Se compone de un conjunto de declaraciones de signaturas de **métodos sin implementar** que especifican un **comportamiento**  específico para una o varias clases con la ventaja de que una clase puede implementar diferentes interfaces.
  + **Generics:** Permiten asignar parámetros a clases, interfaces o métodos de forma que sólo admitan tipos de objetos ya especificados. El siguiente es un ejemplo de implementación de generics:



*(slideshare, 2012)*

* **Tipos de Datos Abstractos:** Se diferencian de los tipos de datos en el sentido de que los datos abstractos son especificados de manera precisas y diseñados independientemente de cualquier implementación, es decir que pueden ser implementados en cualquier lenguaje de programación ya que su forma de definirlos proporciona la información necesaria para hacerlo. El siguiente es un ejemplo de cómo se representa un tipo de dato abstracto de manera formal:

****

*Tomado del Libro J. Villalobos en [Villalobos, 1996].*

* **Pruebas Unitarias:** Una prueba unitaria es un mecanismo que permite comprobar la eficacia de determinado software. Esto se lleva a cabo creando diferentes casos de prueba que, al ejecutarlos en el programa, corroboran el correcto funcionamiento del mismo. El siguiente es un diseño formal de pruebas unitarias:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| Tablero | testTraducirNotacion() | escenario1 | jugada =a7a6  convertida =6858 | Verdadero. La notación se tradujo en coordenada de fila y columna |
| Tablero | testMoverFicha() | escenario1 | jugada= g8h6 | Verdadero. La ficha se movió a la casilla de llegada. |
| Tablero | testNotacionInvalidaExcepcion() | escenario1 | jugada= sdfsdfsd | Verdadero. El mensaje de excepción corresponde al de notación invalida |

*Basado en el Libro J. Villalobos en [Villalobos, 1996].*

TADS VA EN ETAPA 6.

**3. Búsqueda de soluciones creativas:**

|  |
| --- |
| **TAD Hash Table** |
|  |
| X es el conjunto domino de los valores con un tipo de dato particular  U es el universo de llaves |
| * CrearHashTable(entero) -> <Hash Table> * Insertar (U, X) -> <Hash Table> * Buscar (U) -> <Hash Table, X> * Eliminar (U) -> <Hash Table> |

|  |
| --- |
| **CrearHashTable( )** |
| \*Crea un nuevo hash table del tamaño indicado como parámetro.  {pre: TRUE}  {post: hashTable. Se agregó un elemento a la tabla en la posición indicada por la función hash} |

|  |
| --- |
| **Insertar (U, X)** |
| \*Inserta un nuevo elemento en la tabla en la posición indicada por la función hash.  {pre: }  {post: hashTable. Se agregó un elemento a la tabla en la posición indicada por la función hash} |

|  |
| --- |
| **Buscar (U)** |
| \*Busca el elemento en la posición U (llave) de la hash table y lo retorna.  {pre: }  {post: hashTable. Se agregó un elemento a la tabla en la posición indicada por la función hash} |

|  |
| --- |
| **Eliminar (U)** |
| \*Elimina el elemento en la posición U (llave) de la hash table.  {pre: }  {post: hashTable. Se eliminó el elemento de la tabla hash} |

|  |
| --- |
| **EstáVacío ( )** |
| \*Retorna un valor de verdad que indica si la tabla hash está vacía o no.  {pre: }  {post: Determina si la tabla está vacía o no} |
| **TAD Lista** |
| , ,,… , |
| Dónde X es un tipo de dato no primitivo |
| * CrearLista: < > <List> * Insertar (Índice, X) <͢List> * Eliminar (Índice) <List> * EstáVacía ( ) <List> |

|  |
| --- |
| **crearLista ( )** |
| \*Crea una lista sin ningún elemento.  {pre: }  {post: Crea una lista vacía} |

|  |
| --- |
| **agregar (índice, ítem)** |
| \*Agrega el ítem en la posición indicada por el índice.  {pre: }  {post: Inserta el ítem en la posición de la lista indicada por el índice  //Los ítems en la posición índice+1 son movidos una posición a la derecha} |

|  |
| --- |
| **Eliminar (índice)** |
| \*Elimina el elemento de la lista que esté en el índice que entra como parámetro.  {pre: }  {post: Determina si la tabla está vacía o no} |

|  |
| --- |
| **EstáVacío ( )** |
| \*Retorna un valor de verdad que indica si la tabla hash está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

|  |
| --- |
| **Obtener (índice)** |
| \*Retorna el valor que se encuentra en la posición indicada por el índice.  {pre: }  {post: Se ha encontrado el elemento en la posición del índice} |
| **Tamaño ( )** |
| \*Retorna un número entero que indica el tamaño de la lista  {pre: }  {post: Determina el tamaño de la lista} |

|  |
| --- |
| **TAD Cola** |
| , ,,… , |
| Dónde X es un tipo de dato no primitivo |
| * CrearCola ( ) <Cola> * encolar (Ítem) <͢ Cola > * desencolar ( ) < Cola > * EstáVacía ( ) < Cola > |

|  |
| --- |
| **crearCola ( )** |
| \*Crea una cola sin ningún elemento.  {pre: }  {post: Crea una cola vacía} |

|  |
| --- |
| **encolar (ítem)** |
| \*Agrega el ítem al final de la cola.  {pre: }  {post: Se ha agregado el ítem al final de la cola.} |

|  |
| --- |
| **desencolar ()** |
| \*Elimina el elemento que está en el frente de la cola y lo retorna.  {pre: }  {post: Se ha eliminado el último elemento de la cola} |

|  |
| --- |
| **EstáVacío ( )** |
| \*Retorna un valor de verdad que indica si cola está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

|  |
| --- |
| **TAD Pila** |
| , ,,… , |
| Dónde X es un tipo de dato no primitivo |
| * CrearPila: < > <Pila> * Empilar (Índice, X) <͢Pila> * Eliminar (Índice) <Pila> * EstáVacía ( ) <Pila> |

|  |
| --- |
| **crearPila ( )** |
| \*Crea una lista sin ningún elemento.  {pre: }  {post: Crea una pila vacía} |

|  |
| --- |
| **push (ítem)** |
| \*Agrega el ítem en la cima de la pila  {pre: }  {post: Se ha agregado el ítem en la pila} |

|  |
| --- |
| **pop ()** |
| \*Elimina el elemento que está en la cima de la pila.  {pre: }  {post: Determina si la tabla está vacía o no} |

|  |
| --- |
| **EstáVacío ( )** |
| \*Retorna un valor de verdad que indica si la pila está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

|  |
| --- |
| **TAD Heap** |
| , ,, … , |
| * . |
| * CrearMonticulo: < > <Monticulo> * agregar (X) <͢Montículo> * Eliminar (Índice) <Montículo> * EstáVacío ( ) <Montículo> * Maximo ( ) <Montículo> * Ordenar ( ) <Montículo> |

|  |
| --- |
| **crearMonticulo ( )** |
| \*Crea una lista sin ningún elemento.  {pre: }  {post: Crea una pila vacía} |

|  |
| --- |
| **agregar (ítem)** |
| \*Agrega el ítem en la cima de la pila  {pre: }  {post: Se ha agregado el ítem en la pila} |
| **Eliminar ()** |
| \*Elimina el elemento que está en la cima de la pila.  {pre: }  {post: Determina si la tabla está vacía o no} |

|  |
| --- |
| **EstáVacío ( )** |
| \*Retorna un valor de verdad que indica si la pila está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

|  |
| --- |
| **Maximo ( )** |
| \*Retorna un valor de verdad que indica si la pila está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

|  |
| --- |
| **Ordenar ( )** |
| \*Retorna un valor de verdad que indica si la pila está vacía o no.  {pre: }  {post: Determina si la lista está vacía o no} |

**Referencias.**

[**https://josue10.wordpress.com/estructura-de-datos/**](https://josue10.wordpress.com/estructura-de-datos/)

* <http://otrospasos.blogspot.com/2011/10/fifo-y-lifo.html>
* http://www.mundojava.net/interfaces.html?Pg=java\_inicial\_4\_5.html