**INFORME DE INGENIERÍA**

**LABORATORIO UNIDAD 2**

**MARIA CAMILA LENIS RESTREPO**

**JUAN SEBASTIAN PALMA GARCÍA**

**JAVIER ANDRÉS TORRES REYES**

**ALGORITMOS Y ESTRUCTURAS DE DATOS**

**2018-2**

**INFORME DE INGENIERÍA**

**Paso 1: Identificación del problema**

**Definición del problema**

Implementar mejoras para Fortnite usando estructuras de datos vistas en clase.

Justificación

Esto es necesario debido a que el juego tiene una estructura base al ser actualizada constantemente requiere un fortalecimiento de sus algoritmos. Se han recibido varias quejas de jugadores, que reportan una gran inconformidad debido al emparejamiento de búsqueda ya que no consiguen tener una buena conexión en las partida o encuentran jugadores con una habilidad mucho mayor a la de ellos, lo que lleva a una gran desventaja para el que está jugando. Además, se está buscando implementar una exclusividad de plataformas que permita separar los jugadores para que alla una experiencia similar con otros jugadores de la misma plataforma para sentirse en un nivel de juego equilibrado. Por último, se ha decidido tomar la iniciativa de implementar nuevos modos de juego para mantener a la comunidad estable y atraer nuevas multitudes al juego para que este siga creciendo

**Requerimientos funcionales**

|  |  |
| --- | --- |
| **R1** | Emparejar partidas según el nivel de juego. |
| **Descripción** | Emparejar las partidas según el nivel de juego y la geolocalización, de manera que no se afecte el tiempo de espera y los jugadores estén en las condiciones más parecidas posibles. |
| **Entradas** | Jugadores disponibles |
| **Salidas** | Jugadores emparejados, son menos que los disponibles inicialmente. |

|  |  |
| --- | --- |
| **R2** | Categorizar jugadores por su nivel de juego |
| **Descripción** | Categorizar a los jugadores por su nivel de juego, teniendo en cuenta: |
| **Entradas** | \*características\* De los jugadores |
| **Salidas** | Nivel de juego del jugador |

|  |  |
| --- | --- |
| **R3** | Limitar los jugadores en modo plataforma |
| **Descripción** | Limitar la unión de jugadores a una partida dependiendo de la plataforma que esté usando (consola, computador o celular). |
| **Entradas** | Plataforma usada |
| **Salidas** | Si puede aplicar o no a la partida seleccionada. |

|  |  |
| --- | --- |
| **R4** | Recoger armas |
| **Descripción** | En modo San Valentín, el jugador puede acumular armas y solo usar la última que haya adquirido hasta que termine sus municiones y use la adquirida antes que esa. |
| **Entradas** | Arma adquirida |
| **Salidas** | Nueva arma por usar |

|  |  |
| --- | --- |
| **R5** | Desaparecer arma actual |
| **Descripción** | Desparecer arma actual cuando sus municiones se terminen. |
| **Entradas** | Municiones del arma actual |
| **Salidas** | Borrar arma y usar la adquirida antes que esa. |

**Paso 2: Recopilación de la información**

**Generics**

En qué consiste? para qué sirven los generics? notacion de generics?

Los Generics en java, consisten en la creación de clases u/o interfaces con para metros flexibles que pueden ser definidos por el programador.

Los Generics son librerías flexibles que tienen la ventaja de poder ser trasladadas de programa en programa sin ninguna dificultad. Estas clases o interfaces genéricas vuelven mucho más robusto un programa ya que limitan el tipo de datos que pueden recibir. También, hace más fácil la lectura del código al especificar que tipo de dato que se está utilizando.

Notación:

< >: llamados en la programación como diamante, esto es utilizado como un indicador para modificar o aplicar el tipo de datos que va a recibir el Generics.

<https://www.baeldung.com/java-generics>

**Estructuras de datos**

*Pilas*

Una Pila es una lista restringida que consiste en apilar objetos pero no permite el acceso libre libre, en lugar utiliza la metodología LIFO( Last In First Out). Esta lista restringida puede almacenar cualquier tipo de dato mediante la operación push().

Operaciones

* push(item): esta operación tiene la función de agregar el valor especificado a la pila.
* empty(): Regresa un valor Booleano que especifica si la pila esta vacia o no
* pop(): Remueve el objeto que está en el tope de la pila
* peek(): Muestra el objeto en el tope de la pila, sin removerlo.

<http://michelletorres.mx/pila-en-java/>

*Colas*

Una Cola es una lista que consiste en agrupar objetos del tipo abstracto que tienen la característica especial en la cual se agregan al final de la lista y solo pueden ser retirados los que están al inicio o fueron insertados primero.

Operaciones

* Insertar(item): tambien conocida como encolar, consiste en agregar el item deseado al final de la cola
* Extraer(): también conocido como Desencolar, consiste en tomar el primer item dentro de la cola y extraerlo/eliminarlo.
* Consultar(): da la información del objeto que esta ubicado al inicio de la cola sin eliminarlo de la misma

<http://michelletorres.mx/colas-en-java/>

**Paso 3: Búsqueda de soluciones creativas**

Emparejamiento de partidas

* Hacer un puntaje ponderado de las estadísticas bases del juego con valores especiales .
  + Muertes, bajas, tiros a la cabeza, promedio de ranking de supervivencia, tiempo promediado de vida por partida
* Agrupar los jugadores con ping similar o geolocalización aproximada y agruparlos utilizando su cantidad de tiempo jugadas para establecer los grupos
* Cuando un jugador cumpla con ciertos rangos en su geolocalización y unos en sus estadísticas básicas, será utilizado un hash para codificar a que Arraylist de jugadores se le va a enviar.

Modo de juego plataforma

* establecer un universo U que contenga todos los jugadores, y que mediante un hash se distribuyan los jugadores de distintas plataformas a diferente ArrayList independientes
* establecer un universo U que contenga todos los jugadores, y que mediante un hash se distribuyan los jugadores de distintas plataformas a diferentes Colas independientes
* establecer un universo U que contenga todos los jugadores, y que mediante un hash se distribuyan los jugadores de distintas plataformas en una clase especificada dentro del proyecto
* Dividir a los jugadores en cinco colas dependiendo de la plataforma que utilicen
* Incluir la plataforma usada en el perfil del jugador
* Implementar un Hashtable con 6 slots donde cada slot pertenece a un modo de juego y en él se van añadiendo los jugadores a una lista enlazada
* Implementar varios arrays que representen las plataformas y dirigir los jugadores a las mismas
* Usar un Array que contenga Arraylists para insertar los jugadores dentro de los mismos, en los cuales cada uno será asignado una plataforma
* Usar un ArrayList que tenga otros ArrayList que separen a los jugadores por plataforma

**Paso 4: Transición de ideas a los diseños preliminares**

Las ideas descartadas con facilidad son la 5,6,7. Por los siguientes motivos, la opción 7 y 6 tienen la desventaja de requerir un valor fijo para poder crear las plataformas que contendrán a los jugadores: En la opción 6 existe la dificultad de existir un número total de jugadores como si se pudiera predecir cuánta gente entrara a cada modo de juego y si se posiciona un valor muy grande se estaría malgastando memoria; El 7 asume que va a existir una cantidad fija de plataformas, lo que limitaría la cantidad de plataforma que se pueden adicionar y así no se tendrá en cuenta el futuro de vida del juego(nuevas plataformas). Por otro lado, la opción 4 queda fuera ya que solo se está implementando un indicador que por su propia cuenta no separaría a los jugadores en distintos modos de juego, y cuando se busquen armar partidas tomaría mucho tiempo juntar a los jugadores ya que sería un arreglo extenso.

* HashTable: En esta idea se va a utilizar un HashTable que sirva como mapa para las plataformas existentes y así separar los jugadores mediante la conversión de la plataforma en una llave para ubicar el nodo que va almacenar al jugador. Dentro del nodo de almacenamiento existe flexibilidad de múltiples clases o estructuras
* Queues: en esta idea se considera utilizar colas para implementar plataformas que reciban la cantidad de jugadores por plataformas. Se considera utilizarlas de manera individual o en conjunto de un HashMap
* ArrayList: Esta idea se piensa aplicar de manera individual o en conjunto con un HashMap en la cual se quiere codificar la plataforma para agrupar a los jugadores en un arraylist.

**Paso 5: Evaluación o selección de la mejor solución (Criterios y selección)**

Criterio A: Capacidad para añadir una plataforma

- [1] No tiene posibilidad de Agregar nuevas plataformas

- [2]Puede agregar plataformas con restricciones

- [3]Puede agregar plataformas sin restricciones

Criterio B: Capacidad para Eliminar Plataformas

* [1] No puede Eliminar plataformas
* [2] Puede eliminar plataformas con restricciones
* [3] Puede eliminar Plataformas

Criterio C: Tiempo requerido para buscar una Plataforma

* [1]Gran demora en la búsqueda de plataformas
* [2]Demora para la búsqueda de plataformas
* [3]Poca demora para buscar Plataformas

Criterio D: Flexibilidad

* [1] No es flexible
* [2] Poca flexibilidad para utilizar en otros proyectos
* [3] Gran flexibilidad(Se puede aplicar a otros proyectos con facilidad)

Criterio E: Obtención de jugadores para el matchmaker:

* [1] Muy lento en la obtención de Jugadores
* [2] Velocidad Intermedia
* [3] Gran Facilidad

Criterio F: Requerimiento de espacio

* [1] Espacio Fijo
* [2]Espacio variable

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ideas | Criterio A | Criterio B | Criterio C | Criterio D | Criterio E | Criterio F | Total |
| 1 | 2 | 3 | 2 | 3 | 2 | 2 | 14 |
| 2 | 2 | 2 | 1 | 2 | 1 | 2 | 10 |
| 3 | 2 | 3 | 3 | 3 | 3 | 1 | 15 |
| 4 | 3 | 2 | 1 | 2 | 1 | 2 | 11 |
| 5 | 3 | 3 | 2 | 1 | 2 | 2 | 13 |
| 6 | 3 | 3 | 2 | 2 | 2 | 2 | 14 |

1. Hashtable con ArrayList
2. HashTable con Queues
3. HashTable con clase
4. Queues individuales
5. HashTable con lista enlazada
6. ArrayList de ArrayLists

La opción que se va Utilizar es la #3 ya que es la ue obtuvo un mayor puntaje a comparación del resto y tiene la característica de ser extremadamente flexible y es extremadamente eficiente para el matchmaker

**Paso 6: Preparación de informes**

**Diseño del diagrama de clases de la solución**

**Diseño de casos de pruebas unitarias**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba 1: | | | | |
| Clase | Método | Escenario | Entrada | Resultado |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bibliografía**

<https://www.processdash.com/static/teamhelp/Topics/Overview.html>

<https://www.xataka.com/videojuegos/gui-competa-fortnite-todo-que-hay-que-saber-para-empezar-jugar-cero>