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 partidas 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 haya 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: habilidad, partidas jugadas, partidas ganadas y el ping.
Entradas	Habilidad, partidas jugadas, partidas ganadas y ping 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

Criterios para poner nivel a un jugador

Sistema ELO:

El ELO es un sistema matemático, elaborado por el profesor Arpad Elo (Profesor de Física de la Universidad de Milwaukee), para la evaluación del rendimiento de los jugadores de ajedrez. Con él se puede saber sin conocer a un jugador cuál es su nivel de juego y permite realizar clasificaciones de los jugadores. Se calcula al comparar el resultado que obtuvo el jugador en una partida con el resultado esperado de acuerdo a la diferencia de ELO con el contrincante.

Información recopilada de: http://www.jordigonzalezboada.com/ajedrez/elo.html

Sistema MMR (League of Legends)

El MOBA (Multiplayer Online Battle Arena) League of Legends se basa en el índice de emparejamiento (MMR por sus siglas en inglés) para clasificar a sus jugadores. Todos los jugadores comienzan con el mismo MMR la primera vez que entran en un modo de juego. Esta cifra aumenta al ganar y baja al perder. Debido a que este número está oculto, el juego usa un sistema para dar a los jugadores una idea aproximada de su nivel. Este se basa en siete ligas (bronce, plata, oro, platino, diamante, maestro y retador) con cinco divisiones cada una, excepto las últimas dos. Información recopilada de: https://support.riotgames.com/hc/es-419/articles/201752954-Gu%C3%ADa-de-Emparejamiento

Sistema de Fornite Tracker Network

Fortnite Tracker Network es un sitio web que clasifica a todos los jugadores de Fortnite. Para determinar la clasificación de un jugador, la página sólo tiene en cuenta el número de partidas ganadas. Sin embargo, para cada jugador también guarda su porcentaje de victorias (partidas ganadas / partidas jugadas), su número de asesinatos y su K/D (asesinatos / muertes). La página también desarrolló un sistema para determinar la habilidad de un jugador llamado TRN, que se actualiza con cada partida, teniendo en cuenta su posición en la misma y el numero de asesinatos que obtuvo.

Información recopilada de: https://fortnitetracker.com/article/23/trn-rating-you

Emparejamiento en Fortnite

En la actualidad, el emparejamiento no se basa en la habilidad del jugador, sino que, una vez que el jugador entra en espera, se añade a la cola del servidor escogido por él (o seleccionado automáticamente, si así lo desea). Una vez hay alrededor de 100 jugadores, la partida empieza. De esta manera, cualquier jugador se podría enfrentar a cualquier otro, sin importar su nivel. Información recopilada de: https://www.quora.com/How-does-Fortnite-matchmaking-work

Emparejamiento en otro juego (League of Legends)

El sistema de emparejamiento de League of Legends busca que las partidas sean justas, es decir, que los jugadores tengan aproximadamente un 50% de probabilidades de ganar. Para esto, busca que para cada partida, el MMR de sus jugadores sea el más cercano posible.

Latencia y desempeño del juego:

La latencia es el tiempo en que tarda en llegar una información de un lugar a otro, es decir, qué tan alejado se está del servidor del juego: la inmediatez de la conexión. Es medido en milisegundos (ms) y se le llama **ping**, y mide el tiempo que tarda en llegar una información desde un servidor hasta el computador.

Información recopilada de: https://www.xataka.com/basics/que-son-el-ping-y-la-latencia-y-por-que-no-solo-importa-la-velocidad-en-tu-conexion

La velocidad del internet no afecta los *ms* pero puede ayudar o perjudicar en algunos casos, como por ejemplo cuando alguien más está haciendo uso de la conexión a internet en la casa. En este caso se tiene un tiempo estándar del ms (distancia hasta el servidor) pero estos pueden incrementarse porque hay más "tráfico" en el envío y recibimiento de datos.

Información recopilada de: https://boards.las.leagueoflegends.com/es/c/charlas-generales/K9vEUHyj-que-factores-afectan-para-que-tengas-un-ms-alto

Información recopilada de: https://boards.lan.leagueoflegends.com/es/c/ayuda-y-soporte/MwPEdPyX-el-ping-depende-de-mi-internet-o-de-el-servidor

Tiempo de espera promedio para una partida en modo Battle Royale: 30-40 segundos

Modo de juego Plataforma

Actualmente el juego está disponible para cinco plataformas diferentes:

- Playstation 4
- Xbox One
- Nintendo Switch
- Computador Windows o Mac
- Dispositivos móviles iPhone

Información recopilada de: https://www.epicgames.com/fortnite/es-ES/home

Modo de juego San Valentín

Tipos de armas

Rango	**Color**
Gris	Común
Verde	Poco común
Azul	Rara
Morado	Épica
Naranja	Legendaria

Armas conocidas y municiones

Tipo	**Nombre**	**Rango**	**Municiones**
Rifles	M16	Común	30
	M16	Poco común	30
	M16	Raro	30
	SCAR	Épico	30
	SCAR	Legendario	30
	Rifle con visor	Raro	20
	Rifle con visor	Épico	20
	Rifle de asalto a ráfagas	Común	30
	Rifle de asalto a ráfagas	Poco común	30
	Rifle de asalto a ráfagas	Raro	30
	Rifle de asalto a ráfagas	Épico	30
	Rifle de asalto a ráfagas	Legendario	30
	Rifle de asalto con mira térmica	Épico	15
	Rifle de asalto con mira térmica	Legendario	15
SMG	SMG	Común	30
	SMG	Poco común	30
	SMG	Raro	30
	Subfusil compacto	Épico	50
	Subfusil compacto	Legendario	50
Ametralladoras	Minigun	Épico	Depende
	Minigun	Legendario	Depende
	Ametralladora ligera	Raro	100
	Ametralladora ligera	Épico	100
Lanzagranadas	Lanzagranadas	Raro	6
	Lanzagranadas	Épico	6
	Lanzagranadas	Legendario	6
	Lanzacohetes	Raro	1
	Lanzacohetes	Épico	1

	Lanzacohetes	Legendario	1
	Misil teledirigido	Épico	1
	Misil teledirigido	Legendario	1
Escopetas	Escopeta corredera	Poco común	5
	Escopeta corredera	Raro	5
	Escopeta táctica	Común	8
	Escopeta táctica	Poco común	8
	Escopeta táctica	Raro	8
	Escopeta pesada	Épico	7
	Escopeta pesada	Legendario	7
	Escopeta de doble cañón	Épico	2
	Escopeta de doble cañón	Legendario	2
Francotiradores	Francotirador con cerrojo	Raro	1
	Rifle de francotirador de cerrojo	Épico	1
	Rifle de francotirador de cerrojo	Legendario	1
	Francotirador semiautomático	Épico	10
	Francotirador semiautomático	Legendario	10
	Fusil de caza	Poco común	1
	Fusil de caza	Raro	1
	Rifle de francotirador pesado	Épico	1
	Rifle de francotirador pesado	Legendario	1
Pistolas	Pistola	Común	16
	Pistola	Poco común	16
	Pistola	Raro	16
	Pistola con silenciador	Épico	16
	Pistola con silenciador	Legendario	16
	Pistola pesada	Épico	7
	Pistola pesada	Legendario	7
	Revolver	Común	6
	Revolver	Poco común	6
	Revolver	Raro	6
Grandas	Granada	Común	1
	Granada Boogie	Raro	1
	Granada de impulso	Raro	1

Información recopilada de: https://vandal.elespanol.com/guias/guia-fortnite-battle-royale-trucos-y-consejos/armas

Información recopilada de: https://juegosadn.eleconomista.es/guias/guia-fortnite-battle-royale-trucos/consejos-y-trucos/armas/

Información recopilada de: https://db.fortnitetracker.com/weapons

Notas:

-Las armas se consiguen caminando por el mapa, dependiendo de la rareza del arma más difícil será encontrarla.

-Todos los jugadores empiezan la partida con un hacha la cual no cuenta con municiones y no puede desaparecer.

Elementos teóricos

Generics

Los Generics en java, consisten en la creación de clases u/o interfaces con parámetros 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 qué 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.

Información recopilada de: 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, 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 vacía 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.

Información recopilada de: 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): también 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 está ubicado al inicio de la cola sin eliminarlo de la misma

Información recopilada de: http://michelletorres.mx/colas-en-java/

Paso 3: Búsqueda de soluciones creativas

Se realizó una lluvia de ideas para cada mejora que debe implementársele a Fortnite.

Soluciones para emparejamiento de partidas

Para trabajar este problema, se divide en dos subproblemas: asignar un valor que represente la habilidad un jugador y armar partidas de acuerdo con este valor para los jugadores que estén buscando partida en ese momento.

Para el primer problema se tienen las siguientes ideas:

1. *Usar el nivel del jugador.*
2. *Usar la cantidad de partidas jugadas.*
3. *Usar la cantidad total de partidas ganadas.*
4. *Usar el porcentaje de partidas ganadas.*
5. *Usar la posición promedio en la que queda un jugador.*
6. *Usar la cantidad total de asesinatos.*
7. *Usar el promedio de asesinatos por partida.*
8. *Usar el sistema ELO.* El sistema ELO es un número que sirve para predecir el resultado de una partida. Se espera que dos jugadores con igual ELO ganen el mismo número de partidas. Si la diferencia de puntos entre dos jugadores es de 200, se espera que el jugador con más puntos gane el 76% de las partidas contra el primero. El ELO aumenta o decrementa dependiendo del resultado de la partida y del resultado esperado. Si un jugador gana una partida que de acuerdo con el ELO debía ganar, recibe menos puntos que si gana una partida que de acuerdo al ELO debía perder. (Fuente: *https://howlingpixel.com/i-en/Elo_rating_system)*
9. *Usar el sistema TrueSkill.* TrueSkill es un sistema desarrollado por Microsoft Research para clasificar a los jugadores en Xbox Live. Funciona de manera similar al sistema ELO, solo que además de guardar la habilidad de un jugador, guarda un número que representa la incertidumbre en la habilidad de un jugador. Este último aumenta cuando gana una partida que se suponía que iba a perder, o cuando pierde una que debía ganar. (Fuente: https://www.microsoft.com/en-us/research/project/trueskill-ranking-system/)
10. *Usar el sistema TRN.* El sistema TRN es la manera en que la plataforma Fortnite Tracker clasifica a los jugadores. Este sistema asigna un número entre 0 y 5000, siendo 5000 los mejores jugadores. Cuando alguien juega por primera vez, este número vale 1200. El valor se actualiza con cada partida jugada. La página encontró que, en promedio, los jugadores que ganan hacen 7 asesinatos, los segundos 4, los terceros 3 y todos los demás 2. Así cuando alguien gana, y obtiene al menos 7 asesinatos, recibe el 5% de (5000 – puntaje actual), mientras que, si obtuvo menos de 7 asesinatos, recibe le 4% de ese valor. Si el jugador queda en el segundo o tercer lugar, la manera en que se suman puntos funciona igual, aunque ahora con 4 y 3 asesinatos, respectivamente, y recibiendo el 2,5% o 1,5% en lugar del 4% o 5%. Si

el jugador queda en una posición diferente, pierde el 2.5%, aunque si obtuvo al menos 2 asesinatos, solo perderá un 1.5%.

11. *Ponderar la posición promedio con el porcentaje de partidas ganadas y el promedio de asesinatos.*

Por su parte, para el segundo problema están las siguientes ideas:

1. *Formar todas las partidas de 100 jugadores posibles.* Sea n el número de jugadores en cola en un momento dado, se buscan todas las combinaciones posibles de n jugadores en 100 espacios, para finalmente seleccionar los $n/100$ grupos con la menor desviación estándar del valor de emparejamiento de sus jugadores.
2. *Insertar a los jugadores en una cola ordenada.* Los jugadores que entren en la cola se insertarán en una cola ordenada de acuerdo a su valor de emparejamiento, y los grupos de 100 se harán tomando los jugadores en los índices 1-100, luego los del 101-200, etc.
3. *Insertar a los jugadores en una cola de prioridad.* Los jugadores que entren a la cola se añaden a un arreglo. Luego se convierte el arreglo en una cola de prioridad. El jugador en el top de la cola siempre será el que tenga mayor valor de emparejamiento. Así, una partida se formará tomando el jugador en el top y removiéndolo de la cola, hasta que haya 100 jugadores en la partida.
4. *Crear grupos de clasificación más grandes:* De manera similar a como se hace en League of Legends, se forman grupos donde se encuentren todos los jugadores con valor de emparejamiento en cierto rango. Luego, las partidas se formarán tomando en cuenta que sus jugadores pueden pertenecer a un único grupo grande.
5. *Tener en cuenta la diferencia entre el jugador de mayor y de menor Ve.* Al recorrer los jugadores que están esperando, si la diferencia de Ve del jugador actual con el mejor y peor jugador añadidos por el momento no es mayor a una constante, el jugador se añade a la partida.
6. *Llevar control del jugador con mayor desviación.* Primero se toman los primeros 100 jugadores en entrar a la lista. Luego se haya la media de Ve para encontrar el que tiene mayor desviación. Luego se empiezan a recorrer los jugadores restantes. Si la diferencia entre el jugador actual con la media de Ve de los 100 seleccionados es menor a la que tiene el jugador con mayor desviación de la partida, el jugador actual se añade a la partida y el jugador con mayor desviación se remueve.

Hay que tener en cuenta que no en todo momento la cantidad de jugadores buscando partida será múltiplo de 100, por lo que el número máximo de jugadores por partida no se tomaría como 100, sino como $\left\lfloor \frac{n}{\left\lceil \frac{n}{100} \right\rceil} \right\rfloor$, donde n es el número de jugadores buscando partida

Soluciones para modo de juego plataforma

- Dividir a los jugadores en cinco colas dependiendo de la plataforma que usen.
- Incluir la plataforma usada en el perfil del jugador.
- Implementar una Hash Table con cinco slots donde cada slot pertenece a un modo de juego y en él se van añadiendo los jugadores a una lista enlazada.
- 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
- 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

Soluciones para modo de juego San Valentín

Para este modo de juego se tiene en cuenta, como tal, la funcionalidad de acumular y usar las armas recogidas y que solo pueda usarse la última que se ha recogido, de manera que si quiere volver a la anterior deberá acabar con las municiones actuales.

- Implementar un ArrayList en el que vaya añadiendo y eliminando las armas que va recogiendo y usando.
- Implementar una Stack que vaya almacenando las armas recogidas donde la primera arma es un hacha.
- Implementar una lista enlazada donde se van añadiendo en la cabeza las nuevas armas recogidas.
- Implementar una cola en donde se van añadiendo las armas y que el usuario tenga un armaActual como atributo hasta que se terminen las municiones y pase la siguiente arma.
- Implementar un montículo donde la última arma recogida sea “mayor” que las demás y aplicar la función MAX_HEAPIFY

Paso 4: Transición de ideas a los diseños preliminares

Emparejamiento de partidas:

Asignar nivel al jugador:

Las ideas 1, 8, 9 y 10 quedan descartadas por las siguientes razones:

-La idea 1 no tiene en cuenta la skill del jugador, el nivel no dice mucho sobre su habilidad, es más un indicador del tiempo que ha estado juagando.

-La idea 8 está basado en un juego donde es predecible cuando existen chances de ganar la partida o no, como en el ajedrez, pero Fortnite es un juego bastante dinámico que no da lugar a esas inferencias a la hora de jugar una partida, es decir, cualquier cosa puede pasar.

-La idea 9 necesita de datos de partidas anteriores bastante específicos, con los cuales no contamos para esta implementación.

-La idea 10 requiere de una base de datos de más de 5000 jugadores, actualizándose constantemente para conocer el ranking de acuerdo a ella, por lo tanto, está fuera de nuestro alcance de implementación.

Las ideas que quedan son:

1. *Usar la cantidad de partidas jugadas.* El valor que representa la habilidad de un jugador es igual al número total de partidas que ha jugado.
2. *Usar la cantidad total de partidas ganadas.* El valor que representa la habilidad de un jugador es igual al número total de partidas en las que ha quedado en el primer lugar.
3. *Usar el porcentaje de partidas ganadas.* El valor que representa la habilidad de un jugador es igual al número total de partidas en las que ha quedado en el primer lugar dividido entre la cantidad de partidas jugadas.
4. *Usar la posición promedio en la que queda un jugador.* El valor que representa la habilidad de un jugador es igual a la posición promedio en la que queda un jugador en todas sus partidas. Por ejemplo, para un jugador que haya jugado 3 partidas y quedado en las posiciones 25, 26 y 27, su valor será 26.
5. *Usar la cantidad total de asesinatos.* El valor que representa la habilidad de un jugador es igual al número asesinatos que ha realizado.
6. *Usar el promedio de asesinatos por partida.* El valor que representa la habilidad de un jugador es igual al número asesinatos que ha realizado dividido entre la cantidad de partidas jugadas.
7. *Ponderar la posición promedio con el porcentaje de partidas ganadas y el promedio de asesinatos.* El valor que representa la habilidad de un jugador será una ponderación de la posición promedio con el porcentaje de partidas ganadas y el promedio de asesinatos teniendo en cuenta la dificultad de obtener cada una de ellas. Por ejemplo, podrían ser 30%, 50%, 20%, respectivamente.

Emparejar partidas:

La idea 1 queda descartada, pues la cantidad de partidas posibles puede ser demasiado grande. Por ejemplo, si hubiera solo 200 jugadores buscando partida, se podrían armar $\binom{200}{100} \approx 9 \times 10^{58}$ grupos. Por lo tanto, quedan las siguientes ideas:

1. *Ordenar a los jugadores en la lista.* Los jugadores que entren en la cola se insertarán en una cola ordenada de acuerdo a su valor de emparejamiento, y los grupos de 100 se harán tomando los jugadores en los índices 1-100, luego los del 101-200, etc.
2. *Crear grupos de clasificación más grandes:* De manera similar a como se hace en League of Legends, se forman grupos donde se encuentren todos los jugadores con valor de emparejamiento en cierto rango. Luego, las partidas se formarán tomando en cuenta que sus jugadores pueden pertenecer a un único grupo grande.
3. *Tener en cuenta la diferencia entre el jugador de mayor y de menor Ve.* Al recorrer los jugadores que están esperando, si la diferencia de Ve del jugador actual con el mejor y peor jugador añadidos por el momento no es mayor a una constante, el jugador se añade a la partida.
4. *Llevar control del jugador con mayor desviación.* Primero se toman los primeros 100 jugadores en entrar a la lista. Luego se haya la media de Ve para encontrar el que tiene mayor desviación. Luego se empiezan a recorrer los jugadores restantes. Si la diferencia entre el jugador actual con la media de Ve de los 100 seleccionados es menor a la que tiene el jugador con mayor desviación de la partida, el jugador actual se añade a la partida y el jugador con mayor desviación se remueve.

Modo de juego plataforma

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.

- Hash Table: En esta idea se va a utilizar un Hash Table 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 a 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 Hash Map
- ArrayList: Esta idea se piensa aplicar de manera individual o en conjunto con un Hash Map en la cual se quiere codificar la plataforma para agrupar a los jugadores en un Arraylist.
-

Modo juego San Valentín

La idea de implementar una cola queda descartada porque en una cola solo se puede extraer la cabeza, y esta no sería la última arma añadida, entonces no cumple con la funcionalidad requerida. Además, se necesita de otro atributo para conocer el arma actual. Las ideas más factibles son:

- **ArrayList:** En esta idea se tiene un ArrayList donde se van añadiendo las armas y el arma actual siempre será la que se encuentre en la última posición del ArrayList.
- **Stack:** En esta idea la Stack contiene todas las armas, donde la que se encuentre en el top será el arma actual y solo se hará pop si se terminan sus municiones.
- **Lista enlazada:** En esta idea la función de inserción añade en la cabeza la nueva arma, y cuando sus municiones se terminen se elimina el nodo de la cabeza y se reemplaza por el siguiente.
- **Montículo:** En esta idea las armas tienen un numero cada que son añadidas, entonces siempre se cumple la función **HEAP_MAX** donde el arma más reciente será la mayor y estará a la raíz.

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

Emparejamiento de partidas:

Asignar nivel a un jugador:

Criterio A: Datos requeridos: Volumen de datos requeridos para hacer el cálculo.

- [2] Un solo atributo
- [1] Varios atributos

Criterio B: Tiempo de cálculo: Tiempo necesario para calcular alguno de los atributos usados para dar el nivel después de haber jugado una partida.

- [2] O(1)
- [1] O(n) Donde n es la cantidad de partidas jugadas

Criterio C: Veracidad de skill: El ítem a evaluar es lo suficientemente integral para ser considerado la skill del jugador (alguien que juegue muy seguido no necesariamente es mejor que alguien que no lo haga).

- [4] Es sufiente
- [1] No es suficiente

Criterio D: Tiene en cuenta los diferentes estilos de juego: Por ejemplo, jugar a conseguir asesinatos o jugar seguro.

- [3] Tiene en cuenta diferentes estilos
- [1] No tiene en cuenta diferentes estilos

Criterio E: Es necesario jugar muchas partidas para conseguir un valor alto. Es posible ser bueno en un juego sin necesidad de jugarlo muy frecuentemente

- [3] No es necesario
- [1] Es necesario

Solución	Criterio A	Criterio B	Criterio C	Criterio D	Criterio E	Total
1	2	2	1	3	1	9
2	2	2	1	3	1	9
3	2	2	4	3	1	12
4	2	2	4	3	3	14
5	1	1	4	3	3	12
6	1	2	1	1	1	6
7	1	2	4	1	3	11
8	1	1	4	3	3	12

Por lo tanto, el criterio que se tendrá en cuenta es el promedio de partidas ganadas por el jugador.

Además, hay que tener en cuenta que otro criterio muy importante es la latencia que el jugador tenga en ese momento, que si es muy alta puede ser incluso más determinante para el desempeño del jugador en una partida que su habilidad. Por lo tanto, para hallar el valor de emparejamiento para un jugador, se debe ponderar el valor de habilidad con su latencia. Como una latencia menor a 100 no será determinante, por lo que toda latencia menor a 100 se puede asumir como 100; y una latencia mayor a 1000 sería completamente injugable, por lo que se le impedirá el acceso a la cola de emparejamiento a cualquier jugador con latencia mayor a 1000. Así, sean $Ve, Vh\ y\ l$ el valor de emparejamiento, el valor de habilidad y la latencia, respectivamente, definiremos $Ve = Vh * 10 * 0.5 + (1000 - l) * 0.5$, lo que dará un número menor a 1000 (un poco menos, pues la latencia siempre se tomará como al menos 100) y mayor o igual a 0

Matchmaking

Criterio A: La cantidad de jugadores por partida es lo más cercano a 100 posible. Una de las características más importantes de Fortnite es que las partidas son de alrededor de 100 jugadores, por lo que se le da menos puntos a aquellas opciones con las que algunas partidas puedan tener pocos

jugadores (por ejemplo, porque hayan muy pocos que cumplan alguna característica de esa forma de matchmaking).

- [3] Numero lo más cercano a 100 posible.
- [1] Numero no necesariamente cercano a 100

Criterio B: Complejidad temporal. Sea n la cantidad de jugadores esperando partida. Puede ser O(n) si solo hay que recorrer la lista una cantidad de veces, o O(nlogn) si hay que ordenarla antes de recorrerla.

- [2] O(n).
- [1] O(nlogn).

Idea	Criterio A	Criterio B	Total
1	3	1	4
2	1	2	3
3	1	2	3
4	3	2	5

Por lo tanto, para realizar el matchmaking se llevará un control del jugador con mayor desviación para la partida que se esté formando.

Modo de juego plataforma

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

Idea	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

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

Modo de juego San Valentín

Criterio A: Inmediatez para obtener el arma actual: Mide qué tanto proceso se debe hacer para obtener el arma actual en la estructura de datos utilizada.

- [3] Directo: El arma se obtiene directamente con porque se encuentra en la cabeza o en el tope.
- [2] Cálculo: La posición en la que se encuentra el arma debe ser calculada dependiendo del tamaño de la estructura utilizada
- [1] Subproceso: El arma se obtiene directamente después de realizar un subproceso que ubica el arma en la posición deseada (la raíz).

Criterio B: Inmediatez de la eliminación del arma actual: Mide qué tanto proceso debe realizarse para eliminar un arma que se ha quedado sin municiones.

- [3] Directo: Verifica directamente a una posición específica de la estructura (cabeza, tope, raíz) si ya se han agotado las municiones.
- [1] Indirecto: Debe buscar la posición del arma actual y posteriormente verificar sus municiones.

Criterio C: Hacha: Indica el manejo que se le da al hacha en la estructura de datos utilizada. El hacha debe ser la última arma por usar y no tiene municiones

- [3] Final: El hacha de encuentra al final de la estructura (en una hoja, o en el bottom)
- [1] Inicio: El hacha se encuentra en la primera posición de la estructura.

Criterio D: Implementación: Mide que tan pertinentes son las funcionalidades que trae la estructura de datos para el caso.

- [3] Exactas: Las funciones de la estructura cumplen exactamente con lo requerido (insertar al inicio y extraer del inicio cada que se terminen las municiones)
- [2] Falta: Las funciones de la estructura cumplen con lo requerido si se tiene en cuenta algún criterio de orden o de búsqueda para la posición en la que debe añadirse o extraerse elementos.

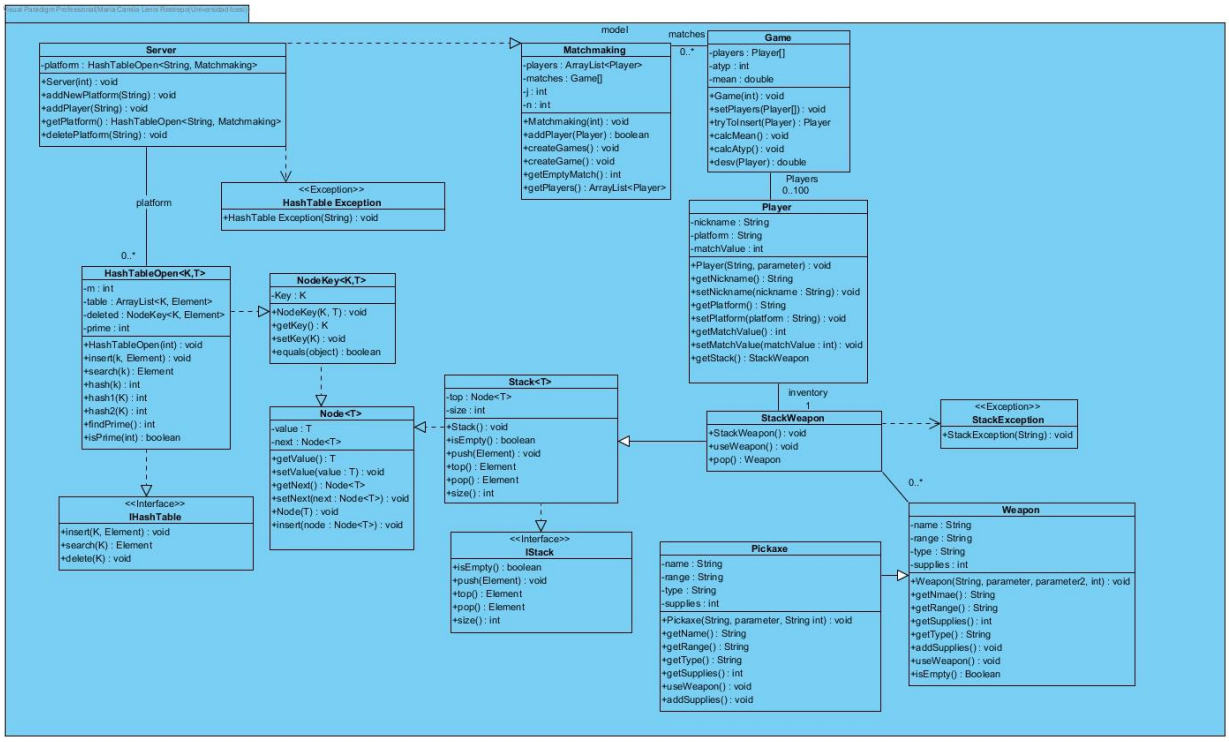
- [1] Sobra: Las funciones de la estructura usada requieren de otro subproceso para mantener el criterio de orden de las armas añadidas y extraidas.

	Criterio A	Criterio B	Criterio C	Criterio D	Total
ArrayList	2	1	1	2	6
Stack	3	3	3	3	12
Lista enlazada	3	3	3	2	11
Montículo	1	3	3	1	8

La estructura de datos a usar será un Stack en el que su primer ítem por defecto será un hacha y esta no podrá hacer pop porque no tiene municiones, por lo tanto, la Stack nunca estará vacía.

Paso 6: Preparación de informes

Diseño del diagrama de clases de la solución



El archivo se encuentra en la carpeta Bibliografía/Diagramas/UML.jpg

Diseño de Tipo de dato abstractos:

Funcionalidad Matchmaking:

TAD Game
Game {(p1, p2, …, pn), atyp, mean}
{inv: n = size((p1, p2, …, pn)) ∧ 1 ≤ n ≤ 100 ∧ (mean = $\frac{\sum_{K=1}^{n} matchValue(p_K)}{n}$ ∧ $\forall p_i, 1 \leq i \leq n : \lvert mean - \text{matchValue}(p_{atyp}) \rvert \geq \lvert mean - \text{matchValue}(p_i) \rvert) \vee \big((\forall i: p_i = null) \rightarrow (atyp = -1 \wedge mean = 0)\big)$}

Operaciones básicas	
• Game	Entero → Game
• setPlayers	Player[] x Game → Game
• tryToInsert	Game x Player → Game x Player

Operaciones

Game(n)
"Crea un game con capacidad para la cantidad de jugadores dada" Pre: n ∈ $\mathbb{Z}$ ∧ 1 ≤ n ≤ 100 Post: game = {(p1, p2, …, pn), -1, 0} ∧ p1…pn ∈ Player ∧ p1…pn = $null$
setPlayers(players,game)
"Modifica la lista de jugadores de la partida por la que llega como parámetro" Pre: game = {(p1, p2, …, pn), …} ∧ players = (pl1, pl2, …, plm) ∧ $m \leq n$ ∧ pl1,pl2,…,plm ∈ Player ∧ p1,p2,…,pm ≠ $null$ Post: game = {(pl1, pl2, …, plm), …}
tryToInsert(p,game)
"Intenta insertar un jugador al juego. Se inserta en el caso de que su desviación sea menor al jugador con mayor desviación, de manera que elimina a este último del juego. Retorna el jugador que queda fuera del juego" Pre: p ∈ Player ∧ p ≠ $null$ ∧ game = {(p1, p2, …, pn), …} ∧ p1…pn ≠ $null$ Post: Si |$mean$ − matchValue(p_{atyp})| > |$mean$ − matchValue(p)|, game = {(p1, p2, …, p_{atyp-1}, p, p_{atyp+1}, …, pn), …} ∧ player = $p_{atyp.}$ De lo contrario, game = {(p1, p2, …, pn), …} ∧ player = p

Diseños de casos de pruebas unitarias

Prueba 1: Verifica que el método setPlayers modifica satisfactoriamente el índice del dato atípico (atyp) dependiendo de la lista de jugadores mandada.				
Clase	Método	Escenario	Entrada	Resultado
Game	setPlayer(Players[]): void	Se ha creado un game con capacidad para 5 jugadores	El array de Players tiene 5 jugadores con los siguientes matchValues respectivamente: 1,2,3,4,10	Atyp=4
Game	setPlayer(Players[]): void	El mismo que el anterior	El array de Players tiene 5 jugadores con los siguientes matchValues respectivamente: 10, 10, 10, 10, 10	Atyp=0
Game	setPlayer(Players[]): void	El mismo que el anterior	El array de Players tiene 5	Atyp=3

			jugadores con los siguientes matchValues respectivamente: 1, 2, 3, 20, 4	
Game	setPlayer(Players[]): void	El mismo que el anterior	El array de Players tiene 5 jugadores con los siguientes matchValues respectivamente: 2, 1, 1, 1, 1	Atyp=0
Game	setPlayer(Players[]): void	El mismo que el anterior	El array de Players tiene 5 jugadores con los siguientes matchValues respectivamente: 10, 15, 20, 25, 30	Atyp=30

Prueba 2: Verifica que el método tryToInsert permite o no la inserción de un jugador nuevo a la partida dependiendo de su matchValue				
Clase	Método	Escenario	Entrada	Resultado
Game	tryToInsert(Player): Player	Un juego con capacidad para 5 jugadores con los siguientes matchValues respectivamente 1, 2, 3, 4, 10	Un jugador con match Value de 5	Un jugador con el matchValue de 10
Game	tryToInsert(Player): Player	El mismo que el anterior.	Un jugador con match Value de 11	Un jugador con el matchValue de 11
Game	tryToInsert(Player): Player	Un juego con capacidad para 5 jugadores con los siguientes matchValues respectivamente 10, 10, 10, 10, 10	Un jugador con match Value de 11	Un jugador con el matchValue de 11
Game	tryToInsert(Player): Player	Un juego con capacidad para 5 jugadores con los siguientes	Un jugador con match Value de 9	Un jugador con el matchValue de 9

		matchValues respectivamente 10, 10, 10, 10, 10		
Game	tryToInsert(Player): Player	Un juego con capacidad para 5 jugadores con los siguientes matchValues respectivamente 10, 15, 20, 25, 30	Un jugador con match Value de 11	Un jugador con el matchValue de 30
Game	tryToInsert(Player): Player	El mismo que el anterior	Un jugador con match Value de 28	Un jugador con el matchValue de 10
Game	tryToInsert(Player): Player	El mismo que el anterior	Un jugador con match Value de 40	Un jugador con el matchValue de 40

TAD Matchmaking
Matchmaking {players=(p1, p2, …, pn), matches=(m1, m2, …, mk), n, j}
{inv: j = m1.players.size, m2.players.size, … = $\left\lceil \frac{n}{\left\lceil \frac{n}{100} \right\rceil} \right\rceil$, $k = size(matches) = \left\lceil \frac{n}{100} \right\rceil$}
Operaciones básicas • Matchmaking Entero → Game • addPlayer Player x Matchmaking → Matchmaking x boolean • createGames Matchmaking → Matchmaking • createGame Matchmaking → Matchmaking • getPlayers Matchmaking → Player[]

Operaciones

Matchmaking(n) "Crea un Matchmaking con capacidad para la cantidad de jugadores dada" Pre: n ∈ $\mathbb{Z}$ ∧ 1 ≤ n Post: game = {(), (m1,m2,…,mk), n, j}
addPlayer(p, mm) "Añade un jugador a la lista de los que están listos, si no se ha rebasado su capacidad" Pre: mm = {players = (p1, p2, …), …} o mm = {(), …} ∧ p ∈ Player ∧ p ≠ $null$ Post: Si size(players) < n, True, mm = {players = (p1, p2, …, p), …} o mm ={(p), …}. De lo contrario, false.
createGames(mm) "Crea las partidas con los jugadores en la lista" Pre: mm = {players=(p1, p2, …, pn), matches=(m1, m2, …, mk), n. j} y ∀m ∈ matches no tiene jugadores Post: mm = {players=(), matches=(m1, m2, …, mk), n, j} y ∀m ∈ matches tiene sus jugadores
createGame(mm)

"Crea una partida con j jugadores" Pre: mm = {players=(p1, p2, …, pn), matches=(m1, m2, …, mi, …, mk), n, j} y mi no tiene jugadores Post: mm = {… , matches=(m1, m2, …, mi, …, mk), n, j} y players tiene j jugadores menos y mi tiene sus jugadores
getPlayers (mm) "Retorna la colección de jugadores listos para jugar." Pre: mm = {players=(p1, p2, …, pn), matches=(m1, m2, …, mi, …, mk), n, j} Post: players=(p1, p2, …, pn)

Prueba 1: Prueba que el método addPlayer añade jugadores al matchmaking mientras no sobrepase su capacidad				
Clase	Método	Escenario	Entrada	Resultado
Matchmaking	addPlayer(Player p): boolean	Se ha creado un matchmaking para capacidad de 10 jugadores y hay 9 jugadores en cola.	Player p tiene Name = "Kwaniss" Plat = XBOX	True
Matchmaking	addPlayer(Player p): boolean	Se ha creado un matchmaking para capacidad de 10 jugadores y hay 10 jugadores en cola.	Player p tiene Name = "Kwaniss" Plat = XBOX	False

Prueba 2: Verifica que el método createGames crea el número de partidas correspondientes con el número de jugadores requerido, de manera que sea lo más cerca posible a 100.				
Clase	Método	Escenario	Entrada	Resultado
Matchmaking	createGames(): void	Se ha creado un Matchmaking con capacidad para 200 jugadores Y se han insertado 200 jugadores en él.	Ninguna	Matches=2 J=100
Matchmaking	createGames(): void	Se ha creado un Matchmaking con capacidad para 250 jugadores Y se han insertado 200 jugadores en él.	Ninguna	Matches=2 J=100
Matchmaking	createGames(): void	Se ha creado un Matchmaking con	Ninguna	Matches=3 J=84

		capacidad para 250 jugadores Y se han insertado 250 jugadores en él.		
Matchmaking	createGames(): void	Se ha creado un Matchmaking con capacidad para 330 jugadores Y se han insertado 330 jugadores en él.	Ninguna	Matches=4 J=83
Matchmaking	createGames(): void	Se ha creado un Matchmaking con capacidad para 70 jugadores Y se han insertado 70 jugadores en él.	Ninguna	Matches=1 J=70

Prueba 3: Verifica que el método createGame, crea un primer juego con los jugadores que están en el match de manera que tenga la capacidad adecuada y además tengan una desviación similar sin importar si fueron insertados primero o después de otros.				
Clase	Método	Escenario	Entrada	Resultado
Matchmaking	createGame(): void	Se ha creado un Matchmaking con capacidad para 200 jugadores los cuales están insertados de la siguiente manera: 100 tienen matchValue de 10 100 tienen matchValue de 20	Ninguna	El primer match es una partida con capacidad para 100 jugadores con mean 10.
Matchmaking	createGame(): void	Se ha creado un Matchmaking con capacidad para 200 jugadores los cuales están insertados de la siguiente manera: Los 100 primero jugadores tienen un matchValue	Ninguna	El primer match acepta hasta un matchValue de 50. Quedan 100 jugadores en el match

		empezando desde 1 hasta 100, los siguientes 100 jugadores están insertados de la misma manera.		
Matchmaking	createGame(): void	Se ha creado un Matchmaking con capacidad para 100 jugadores los cuales están insertados de la siguiente manera: Los 100 jugadores tienen un matchValue de 30	Ninguna	No quedan jugadores en el matchMaking
Matchmaking	createGame(): void	Se ha creado un Matchmaking con capacidad para 300 jugadores los cuales están insertados de la siguiente manera: Los 50 primeros jugadores tienen un matchValue de 30, los siguientes 50 tienen un match value de 60, y se repite hasta completar 300.	Ninguna	El primer match tiene 100 jugadores con mean 30.
Matchmaking	createGame(): void	Se ha creado un Matchmaking con capacidad para 200 jugadores los cuales están insertados de la siguiente manera: Los 30 primeros tiene un matchValue de 20, y los 170 restantes tienen un matchValue de 50	Ninguna	El primer match tiene un mean de 50
Matchmaking	createGame(): void	Se ha creado un Matchmaking con	Ninguna	El primer match es una partida

		capacidad para 180 jugadores los cuales están insertados de la siguiente manera: Los primeros 50 tienen un matchValue de 3, los siguientes 40 tienen un matchvalue de 50, los siguientes 50 tienen un match value de 45, y los siguientes 40 tienen un match value de 20		con 90 jugadores con una mean de 10. Y la otra partida armada con los jugadores restantes tiene un mean de 47

Funcionalidad modo plataforma

TADServer	
Server{ Platform=<HashTable<K,T>>}	
Inv: Es necesario que se reciba un valor n>=6	
Operaciones Básicas ● Server ● addNewPlatform ● addPlayer ● deletePlatform ● getPlatform	 int→Server Server x String → Server Server x Player→Server Server x String→Server Server → HashTableOpen<K,T>>

Operaciones Básicas

Server() “Crea una Clase Servidor” Pre: un n valor >=6 Post: Server={Platform=<HashTableOpen<K, T>}
getPlatform() “Retorna la HashTable Plataforma” Pre: la clase Server debe de ser instanciada Post: <HashTableOpen<K,T>
addNewPlatform() “Crea una Cola dentro del HashTable con una llave que pertenezca al String ingresado”

Pre: HashTableOpen<> size>0 ^ String!=null Post: Se agrega el nuevo queue con su respectivo key al servidor o se lanza una excepción sí no hay espacio disponible en la Tabla Hash
addNewPlayer() “Agrega un Jugador a la cola utilizando su plataforma como llave para asignar una cola Pre: Player: Platform!=null ^ Platform ∈ h(K). Post: Se agrega el Player a la Queue que le corresponde.
deletePlatform() “Se elimina la info dentro del HashTable con el valor del String dado” Pre: String!=null Post: Platform.search(String)==Null

TAD HashTableOpen< K, T>	
HashTableOpen { m=<int>, table=<ArrayList<NodeKey<K,T>>, prime=<int>, deleted=NodeKey<K,T>}	
Inv: m debe ser un int ^ m>=1	
Operaciones Básicas:	
● HashTableOpen	Int→HashTableOpen
● Insert	K(key) x T(Value) x ArrayList<K,T>→ArrayList<K,T>
● Search	K(key) x ArrayList<K,T>→T (Value)
● Delete	K(key) x ArrayList<K,T>→ArrayList<K,T>
● Hash	K(key) x int→ int
● Hash1	K(key) → int
● Hash2	K(key) → int
● findPrime	→ int
● isPrime	itn → boolean

Operaciones Basicas

HashTableOpen() “Se crea la clase HashTableOpen” Pre: m ∈ N Post: HashTableOpen {m=<int>, table=<ArrayList<NodeKey<K,T>>, prime=<int>, deleted=NodeKey<K,T>}
Insert() “Este metodo adiciona un elemento T en la llave K asignada Pre: K∈ N ∧ K<table.size() ∧ hay espacios disponibles en table Post: Table={a,b,c,...,nm-1, m}
Search() “Este metodo busca y retorna el valor T que corresponde a la llave K Pre: Table!= null ⋀ K∈ N ∧ K<Table.Size() Post: NodeKey<T>

Delete()
“Este metodo recibe una llave K y elimina el NodeKey <T> que tiene esa llave
Pre: K∈ N ∧ K!=null
Post: Table(K)=deleted

Hash()
“Este metodo convierte una llave K y un int en un valor int
Pre: int>=0 ∧ K!=null
Post: Se genera un int que sirve como llave para el HashTable

hash1()
“Este Método le saca un valor hash a la llave (Método de Java)”
Pre:K!=null
Post: |Int|

hash2()
“Este metodo le saca un valor hash a un primo con la resta del hash code de la llave modulo del primo”
Pre: K!=null
Post: Int

findPrime()
Este método busca un valor primo
Pre: True
Post: int

isPrime()
“Este método verifica si int es primo o no
Pre: int>2
Post:True si int es primo;False si int no es primo

TAD Player

Player {
nickname=<name>platform=<platform>,platformMode=<platformmode>,matchValue=<matchvalue>, ability=<ability>,ping=<ping>, playedGames=<playedgames>, wonGames=<wongames>}

Inv: nickname != null ∧ platform ∈ h(Keys)

Operaciones Basicas

- Player() String x String → Player
- calcAbility Player → Player
- calcMatchValue Player→ Player
- getNickname() Player→ String
- setNickname() String x Player → Player
- isPlatformMode() Player → boolean
- setPlatformMode() boolean x Player→ Player
- getPlatform() Player→ String
- setPlatform() String x Player→ Player
- getMatchValue() Player→ double

• setMatchValue()	double x Player→ Player
• getAbility()	Player→double
• setAbility()	double x Player→ Player
• getPing()	Player→ int
• setPing()	int x Player→ Player
• getPlayedGames()	Player → int
• setPlayedGames()	int x Player→ Player
• getWonGames()	Player·→ int
• setWonGames()	int x Player → Player

Operaciones Basicas

Player() Este metodo es el constructor de la clase Player Pre: String name!= null ∧ String platform !=null ∧ platform ∈ h(K) Post: Player
calcAbility() Este metodo calcula la habilidad del jugador Pre: True Post: Calcula el valor de Ability
calcMatchValue() Este metodo calcula el valor de partida del jugador Pre: True Post: Calcula el valor de MatchValue
getNickname() Este metodo retorna el String del NickName Pre: True Post:
setNickname() Este metodo define la variable nickname Pre: String!= null Post:cambia el valor de nickname
getPlatform() Este metodo retorna el valor de la variable Platform Pre: True Post:
setPlatform() Este metodo define el valor de la variable Platform Pre: string nueo !=null Post:cambia el valor de platform
getMatchValue() Este retonra el valor de la variable matchValue

Pre:True Post:
setMatchValue() Este metodo define el valor de la variable matchValue Pre: New double !=null Post:cambia el valor de matchValue
getAbility Retorna el valor de la variable ability Pre: True Post:
setAbility Modifica el valor de ability Pre: new double !=null ∧ 0<=double<=1 Post:cambia el valor de cambia el valor de ability
getPing Este método retorna el valor de la variable ping Pre: True Post:
setPing Este Metodo modifica el valor de ping Pre: new Int !=null ∧ 0<int Post:cambia el valor de ping
getPlayedGames Da el valor de la variable played games Pre: True Post:
setPlayedGames Modifica el valor de la variable playedGames Pre: new int !=null ∧ int>=0 Post:cambia el valor de playedGames
getWonGames Este metodo retorna el valor de la variable wonGames Pre: True Post:
setWonGames Este metodo modifica el valor de la variable WonGames Pre: new Int!= null ∧ int>=0 Post:cambia el valor de wonGames

Diseño de casos de pruebas unitarias:

Prueba 1: Prueba que el método addNewPlatform agrega plataformas correctamente a la HashTable				
Clase	Metodo	Escenario	Valores de entrada	Salida
Server	AddNewPlatform()	Se crea el servidor y se agregan las 6 plataformas y se verifica si se adiciono la cola con la llave "Switch"	Ninguno	True, se ha adicionado con exito la cola
Server	AddNewPlatform()	Se crea el servidor junto las 6 plataformas y se agrega una nueva Queue con la llave "Sega"	"Sega"	True
Server	AddNewPlatform()	Se crea el servidor con las 6 plataformas	Se agrega la queue, "Sega", "GameBoy", "NES", "Failure"	Se agregan más plataformas de las que se puede tener dentro del arreglo y se verifica que la excepción que se lanza sea del Tipo HashTable Exception

Prueba 2: Prueba que el método addPlayer agrega correctamente jugadores a los Queues dentro del HashTable				
Clase	Metodo	Escenario	Valores de entrada	Salida
Server	addPlayer()	Se crea el servidor y se agregan las 6 plataformas, se agrega un nuevo jugador y se agrega a la cola con la llave "Xbox" y se agrega otro a la cola de "Switch"; Se verifica el tamaño de las colas	Player("Ninja"," Xbox"), Player("Rambo", "Switch")	True, las dos colas tienen el tamaño de 1.

Server	addPlayer()	Se crea el servidor y 6 plataformas, se agregan 3 jugadores a una misma queue y se evalua su tamaño	Player(“Jatr9908”, “Pc”), Player(“Millionhat”, “Pc”), Player(“KungFumasta”, “Pc”)	True, la cola del tipo Pc debe de tener un tamaño equivalente a 3
Server	addPlayer()	Se crea el servidor y las 6 plataformas, se agregan dos jugadores con una plataforma errónea o inexistente y se verifica si fueron agregados	Player(“pinkyFlower”, “SuperNintendo”) Player(“TripaDeAcero”, “Atari”)	Se evalua que se lanze una excepción del tipo nullpointer ya que no existe ninguno de las 2 paltaformas

Prueba 3: Prueba que el método deletePlatform elimina correctamente llaves en el HashTable.				
Clase	Metodo	Escenario	Valores de entrada	Salida
Server	deletePlatform()	Se crea un Servidor con las seis plateformas basicas: Pc,PlayStation,Xbox,Switch,Mobile, todos.	“Pc” String	Se verifica que no haya nada en relaciona al key de Pc
Server	deletePlatform()	Se crea un Servidor con las seis plateformas basicas: Pc,PlayStation,Xbox,Switch,Mobile, todos. Se agrega la plataforma Sega	“Sega”	Se verifica si el valor establecido para Sega es Nulo dentro de la tabla Hash
Server	deletePlatform()	Se crea un Servidor con las seis plateformas basicas: Pc,PlayStation,Xbox,Switch,Mobile, todos.	“Sega”	Se verifica que se lanze una excepcion del tipo null pointer ya que no existe esa plataforma

StackWeapon

TAD Weapon
Weapon {Name = <name>, Type = <type>, Range = <range>, Supplies = <supplies>}

{inv: $supplies \geq 0$ }
Operaciones básicas • Weapon Texto x Texto x Texto x Entero → Weapon • getSupplies Weapon → Entero • getName Weapon → Texto • getRange Weapon → Texto • getType Weapon → Texto • addSupplies Weapon x Entero → Weapon • useWeapon Weapon → Weapon • isEmpty Weapon → booleano

Operaciones

Weapon(name, range, type, supplies) "Crea una Weapon con los parámetros pasados" Pre: $name \in Texto \land range \in Texto \land type \in Texto \land supplies \in \mathbb{Z} \land supplies \geq 0$ Post: weapon = {Name = name, Type = type, Range = range, Supplies = supplies}
getSupplies(weapon) "Retorna la munición actual del arma" Pre: weapon = {…, Supplies = <supplies>} Post: <supplies>
getName(weapon) "Retorna el nombre del arma" Pre: weapon = {…, Name = <name>, …} Post: <name>
getRange(weapon) "Retorna el rango del arma" Pre: weapon = {…, Range = <range>, …} Post: <range>
getType(weapon) "Retorna el tipo del arma" Pre: weapon = {…, Type = <type>, …} Post: <type>
addSupplies(weapon, n) "Añade munición al arma" Pre: weapon = {…, Supplies = <supplies>, …} $\land n \in \mathbb{Z} \land n \geq 1$ Post: weapon = {…, Supplies = <supplies>+1, …}
useWeapon(weapon) "Dispara una munición del arma" Pre: weapon = {…, Supplies = <supplies>, …} Post: weapon = {…, Supplies = <supplies>-1, …} si <supplies> > 0 , weapon = {…, Supplies = <supplies>, …} de lo contrario
isEmpty(weapon) "Verifica si el arma está vacía" Pre: weapon = {…, Supplies = <supplies>, …} Post: weapon = True si <supplies> $= 0$, False de lo contrario
TAD Pickaxe
Pickaxe {Name = <name>, Type = <type>, Range = <range>, Supplies = <supplies>}
{inv: $Name =$ Pickaxe $\land Range =$ Basic $\land Type =$ Default $\land Supplies = 1$}
Operaciones básicas • Pickaxe - → Pickaxe

- getSupplies Pickaxe → Entero
- getName Pickaxe → Texto
- getRange Pickaxe → Texto
- getType Pickaxe → Texto
- useWeapon Pickaxe → Pickaxe
- isEmpty Pickaxe → booleano

Operaciones

Pickaxe()
"Crea una Pickaxe"
Pre: $True$
Post: Pickaxe = {Name = "Pickaxe", Type = "Default", Range = "Basic", Supplies = 1}

getSupplies(Pickaxe)
"Retorna la munición actual del hacha"
Pre: Pickaxe = {…, Supplies = <supplies>}
Post: <supplies>

getName(Pickaxe)
"Retorna el nombre del hacha"
Pre: Pickaxe = {…, Name = <name>, …}
Post: <name>

getRange(Pickaxe)
"Retorna el rango del hacha"
Pre: Pickaxe = {…, Range = <range>, …}
Post: <range>

getType(Pickaxe)
"Retorna el tipo del hacha"
Pre: Pickaxe = {…, Type = <type>, …}
Post: <type>

useWeapon(Pickaxe)
"Usa el hacha"
Pre: Pickaxe = {Name = <name>, Type = <type>, Range = <range>, Supplies = <supplies>}
Post: Pickaxe = {Name = <name>, Type = <type>, Range = <range>, Supplies = <supplies>}

isEmpty(Pickaxe)
"Verifica si el arma está vacía"
Pre: Pickaxe = {Name = <name>, Type = <type>, Range = <range>, Supplies = <supplies>}
Post: Pickaxe = True si <supplies> $= 0$, False de lo contrario

TAD StackWeapon

StackWeapon = {(a1, a2, …, an), top}

{inv: $a1 \in Pickaxe \wedge a2 \dots an \in Weapon \wedge size(StackWeapon) \geq 1 \wedge top = an \wedge isEmpty(top) = false$

Operaciones básicas
- StackWeapon - → StackWeapon
- push StackWeapon x Weapon → StackWeapon
- top StackWeapon → Weapon ∨ Pickaxe
- pop StackWeapon → StackWeapon x Weapon
- size StackWeapon→ Entero
- useWeapon StackWeapon → StackWeapon

Operaciones

StackWeapon() "Crea una StackWeapon" Pre: $True$ Post: StackWeapon = {a1}
push(s,w) "Añade un arma al StackWeapon" Pre: s = {a1, a2, …, an}, w ∈ Weapon Post: s = {a1, a2, …, an, w}
top(s) "Retorna el elemento en el top del StackWeapon" Pre: s = {a1, a2, …, an} ∨ s = {a1} Post: Weapon an si s = {a1, a2, …, an}, Pickaxe a1 si s = {a1}
pop(s) "Elimina la última arma añadida al StackWeapon y la retorna" Pre: s ≠ {a1}, es decir, s = {a1, a2, …, an} Post: s = {a1, a2, …, an-1}, Weapon an
size(s) "Retorna la cantidad de armas en el StackWeapon" Pre: s = {a1, a2, …, an} Post: n ∈ ℤ
useWeapon(s) "Usa la última arma añadida al StackWeapon, y la elimina si se queda sin munición y no es un hacha" Pre: s = {a1, a2, …, an} ∨ s = {a1} Post: s = {a1} si s = {a1}. De lo contrario, si getSupplies(an) = 1, s = {a1, a2, …, an-1}; de lo contrario s = {a1, a2, …, an={…, Supplies = <supplies>-1}}

Diseño de casos de pruebas unitarias:

Prueba 1: Añade elementos correctamente a la Stack				
Clase	Método	Escenario	Entrada	Resultado
StackWeapon	Push(Weapon w)	Se ha creado un StackWeapon	W tiene Nombre= "M16" Range="Common" Type="Rifle" Supplies=30	El elemento del top es un Rifle común llamado M16 con 30 municiones. La Stack es de tamaño 2.
StackWeapon	Push(Weapon w)	Se ha creado un StackWeapon que tiene 2 armas, el hacha que viene por defecto y la siguiente arma: Nombre= "M16" Range="Common" Type="Rifle" Supplies=30	W tiene Nombre= "SMG" Range="Rare" Type="SMG" Supplies=30	El elemento del top es un Rifle raro llamado SMG con 30 municiones. La Stack es de tamaño 3.

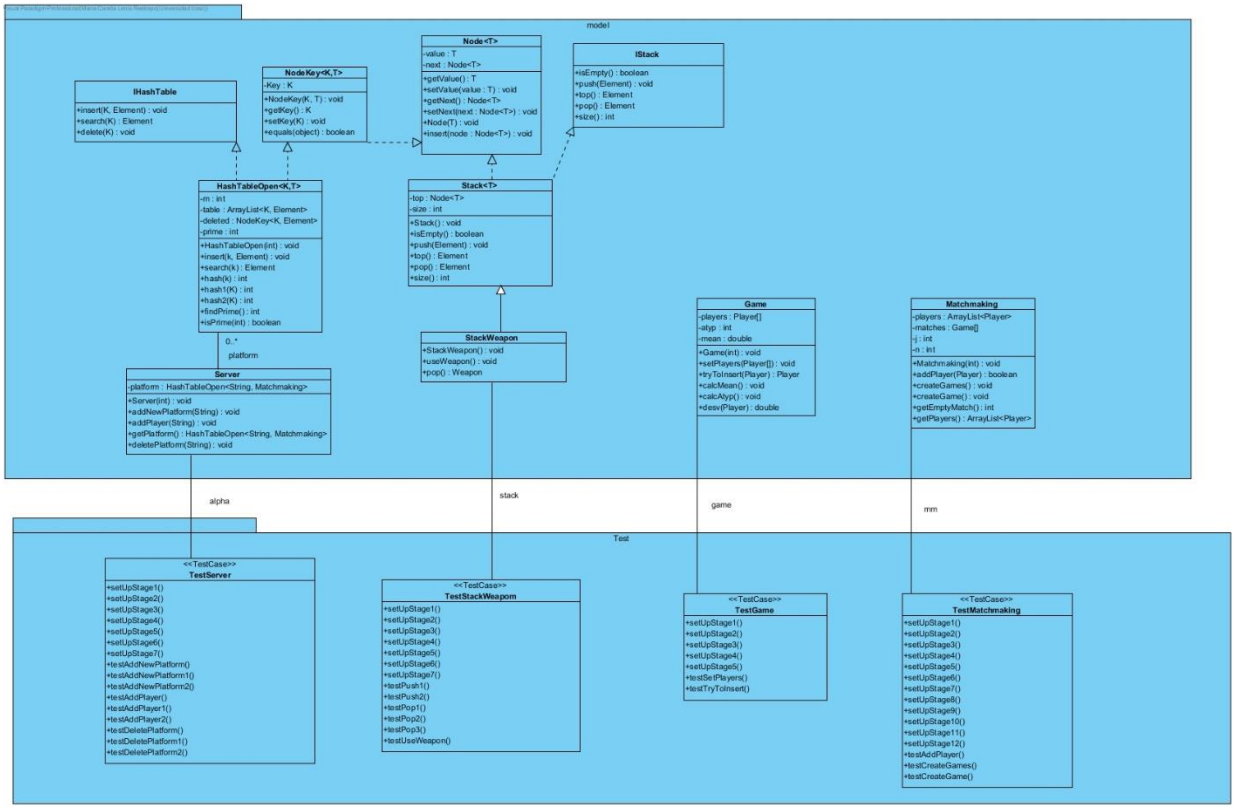
Prueba 2: Elimina elementos satisfactoriamente de la Stack				
Clase	Método	Escenario	Entrada	Resultado
StackWeapon	+Weapon Pop()	Se ha creado un StackWeapon	Ninguna	Lanza una excepción porque no se puede eliminar el hacha de la Stack.
StackWeapon	+Weapon Pop()	Se ha creado un StackWeapon y se añadió la siguiente arma: Nombre= “M16” Range=”Common” Type=”Rifle” Supplies=30	Ninguna	El elemento del top de la Stack es el hacha y la Stack es de tamaño 1.
StackWeapon	+Weapon Pop()	Se ha creado un StackWeapon y se añadió la siguiente arma: Nombre= “M16” Range=”Common” Type=”Rifle” Supplies=30 Y luego la siguiente arma: Nombre= “SMG” Range=”Rare” Type=”SMG” Supplies=30	Ninguna	El elemento del top de la Stack es el M16 y la Stack es de tamaño 2.

Prueba 2: El método use weapon decrementa las municiones del arma y la elimina del Stack cuando se quede sin municiones.				
Clase	Método	Escenario	Entrada	Resultado
StackWeapon	+ void useWeapon()	Se ha creado una StackWeapon	Ninguna	El elemento del top es el hacha por lo tanto sus municiones siguen siendo uno y la Stack es de tamaño 1
StackWeapon	+ void useWeapon()	Se ha creado una StackWeapon. Se le ha añadido la siguiente arma:	Ninguna	El elemento del top es el M16, al que le queda una munición.

		Nombre= “M16” Range=”Common” Type=”Rifle” Supplies=30 Y se usa 29 veces		
StackWeapon	+ void useWeapon()	Se ha creado una StackWeapon. Se le ha añadido la siguiente arma: Nombre= “M16” Range=”Common” Type=”Rifle” Supplies=30 Y se usa 30 veces	Ninguna	El elemento del top es el hacha, la Stack es de tamaño 1.
StackWeapon	+ void useWeapon()	Se ha creado una StackWeapon. Se le ha añadido la siguiente arma: Nombre= “M16” Range=”Common” Type=”Rifle” Supplies=30 Y luego la siguiente arma: Nombre= “SMG” Range=”Rare” Type=”SMG” Supplies=30 Y se usa 29 veces el arma del top.	Ninguna	El elemento del top es el SMG y su atributo supplies=1
StackWeapon	+ void useWeapon()	El mismo que el anterior, pero el arma del top se usa 30 veces	Ninguna	El elemento del top es el M16 y su elemento supplies es 30
StackWeapon	+ void useWeapon()	El mismo que el anterior, pero el arma del top se usa 29 veces Y Luego se le añade un arma	Ninguna	El elemento del top es el M16 y su elemento supplies es 30

		Nombre= “Rocket Launcher” Range=”Rare” Type=”Granade Launcher” Supplies=1 Y se usa el arma 2 veces.		

Diagrama de pruebas unitarias



El archivo se encuentra en la carpeta Bibliografía/Diagramas/Tests.jpg

Bibliografía

Anónimo. (Desconocido). “El elo en Ajedrez”. Tomado de: http://www.jordigonzalezboada.com/ajedrez/elo.html

Anónimo. (29 de Agosto, 2017) “¿El ping depende de mí Internet o del Servidor?”. Tomado de: https://boards.lan.leagueoflegends.com/es/c/ayuda-y-soporte/MwPEdPyX-el-ping-depende-de-mi-internet-o-de-el-servidor

Anónimo.(18 de septiembre 2018). “Guía de Emparejamiento”. Tomado de: https://support.riotgames.com/hc/es-419/articles/201752954-Gu%C3%ADa-de-Emparejamiento

Anónimo. (9 de mayo 2018). “How does Fortnite matchmaking work?”. Tomado de: https://www.quora.com/How-does-Fortnite-matchmaking-work

Anónimo. (15 de septiembre 2017). “¿Qué factores afectan para que tenga un MS alto?”. Tomado de: https://boards.las.leagueoflegends.com/es/c/charlas-generales/K9vEUHyj-que-factores-afectan-para-que-tengas-un-ms-alto

Anónimo.(22 de Mayo 2017). “Que son el ping y la latencia y porqué no solo la velocidad de tu conexión". Tomado de: https://www.xataka.com/basics/que-son-el-ping-y-la-latencia-y-por-que-no-solo-importa-la-velocidad-en-tu-conexion

Anónimo. (3 de Junio, 2015) “TDA Pila en java”. Tomado de: http://michelletorres.mx/pila-en-java/

Anónimo. (18 de Agosto, 2018). “The Basics of Generics”. Tomado de: https://www.baeldung.com/java-generics

Anónimo.(20 de Septiembre 2018). “TRN Rating & You”. Tomado de: https://fortnitetracker.com/article/23/trn-rating-you

Anónimo.(13 de Agosto 2018). “Todas las armas de Fortnite: Battle Royale”. Tomado de: https://juegosadn.eleconomista.es/guias/guia-fortnite-battle-royale-trucos/consejos-y-trucos/armas/

Anónimo. (Desconocido). “Weapons”. Tomado de: https://db.fortnitetracker.com/weapons

Caballero, Daniel A. (27 de Agosto 2014). “Colas en Java”. Tomado de: http://michelletorres.mx/colas-en-java/

Frankie M.(3 de Agosto 2018). “La guía completa de ‘Fortnite’: todo lo que hay que saber para empezar a jugar desde cero”. Tomado de: https://www.xataka.com/videojuegos/gui-competa-fortnite-todo-que-hay-que-saber-para-empezar-jugar-cero

Rebolledo, Cesar. (20 de Agosto 2018). “Fortnite Battle Royale: TODAS las armas y sus estadísticas (ACTUALIZADO)”. Tomado de: https://vandal.elespanol.com/guias/guia-fortnite-battle-royale-trucos-y-consejos/armas