PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Implementación de un algoritmo metaheurístico Cuckoo Search, para sistemas de premiación de juegos

Tesis para obtener el título profesional de Ingeniero Informático

AUTOR

Lucas Augusto Castañeda Quiñones

ASESOR:

Mag. Rony Cueva Moscoso

Lima, Marzo, 2022

Resumen

El presente proyecto de fin de carrera propone implementar un algoritmo metaheurístico, *cuckoo search*, en el proceso de obtención de recompensas de juegos Gacha. El foco y objetivo de este estudio es el poder encontrar un equilibrio entre la satisfacción del usuario y el beneficio de la empresa, por lo cual se utilizaron dos tipos de usuario quienes abarcan las características de tiempo empleado en el juego y cuánto monto han invertido en éste.

Para ello, se propuso una función objetivo en la cual abarca las variables relacionadas al usuario y la empresa, luego se adaptó el algoritmo propuesto al contexto planteado. Finalmente se implementó y aplicó en un prototipo de juego donde se compara el funcionamiento y desempeño de éste junto a un simulador; además de poder visualizar y simular el contexto de estar utilizando/jugando un juego Gacha. De los resultados, se pudo verificar un desempeño del algoritmo elegido frente al simulador. Con ello se logra cumplir con el objetivo inicial de poder equilibrar los valores representativos del usuario y el beneficio de la empresa. La meta propuesta es poder demostrar que el uso del cuckoo search en estos juegos es posible y en un futuro poder mejorarlo para su uso en estos tipos de juegos.

Dedicatoria

Dedico este proyecto de tesis a todas las personas que hicieron posible que mi anhelado sueño se vuelva realidad.

A mi querido asesor Rony Cueva que confío en mi innovador proyecto y que con sus invalorables conocimientos me guío y acompañó hasta el final. De igual manera a mis profesores por sus duras y constructivas críticas que ayudaron a formarlo.

A mis queridos amigos y compañeros que fueron un gran apoyo para poder continuar.

A mi querido padre, hoy en el cielo, que al igual que él, sigo su camino por esta profesión.

A mi entrañable hermana, que es mi fuente de inspiración.

Y una admirable mujer, que, con una frase histórica, logró esta gran hazaña… mi amada Madre.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades

1.1 Problemática

El mercado de los videojuegos ha sido controlado mayormente por las consolas, pero en los últimos años, las plataformas de videojuegos están evolucionando. Según Batchelor, se ha generado 134.9 mil millones de dólares en el transcurso del año 2018 (ver Figura 1). Entre estas cifras, 63.2 mil millones de dólares lo proporcionó el sector móvil (47%) repartido entre juegos de Smartphone y Tablet. Gracias a estas cifras, los juegos móviles en sus diversos estilos, son el segmento con el crecimiento más rápido dentro de la industria, provocando que cada vez más compañías y grupos independientes se enfocan en este sector; un claro ejemplo es Sony con su juego *Fate/Grand Order* el cual desde su debut (Agosto del 2015) ha recaudado un estimado de 3 mil millones de dólares a través de la *App Store* y *Google Play* mundialmente.

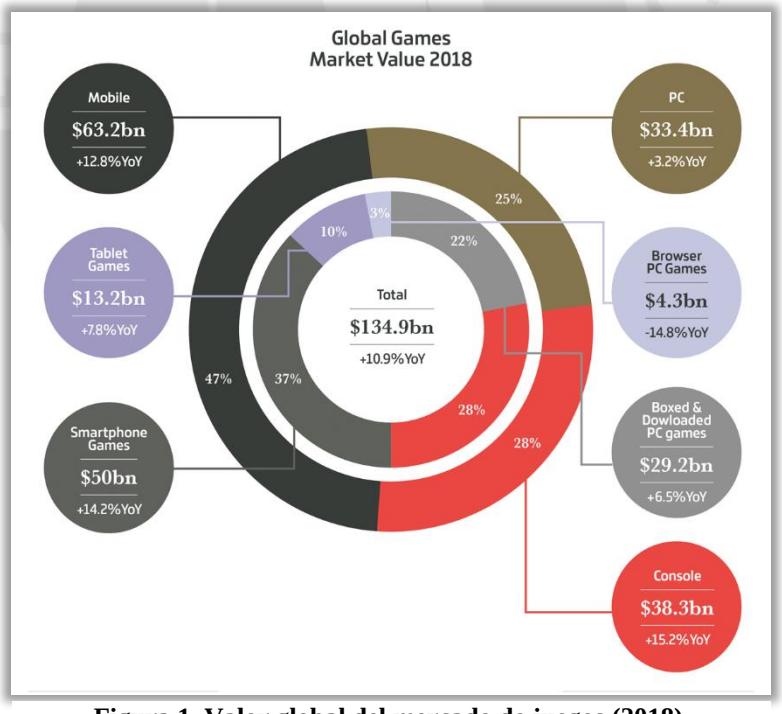


Figura 1. Valor global del mercado de juegos (2018).
Fuente: GamesIndustry

Como es en el caso de los juegos de consola, los juegos móviles tienen distintos géneros, tipos y mecánicas. Entre estos, los videojuegos gratuitos (*Free to Play*, F2P) son los más

atrayentes para los jugadores. Dentro de este tipo se encuentran los juegos *"Gacha"* (ver Figura 2).

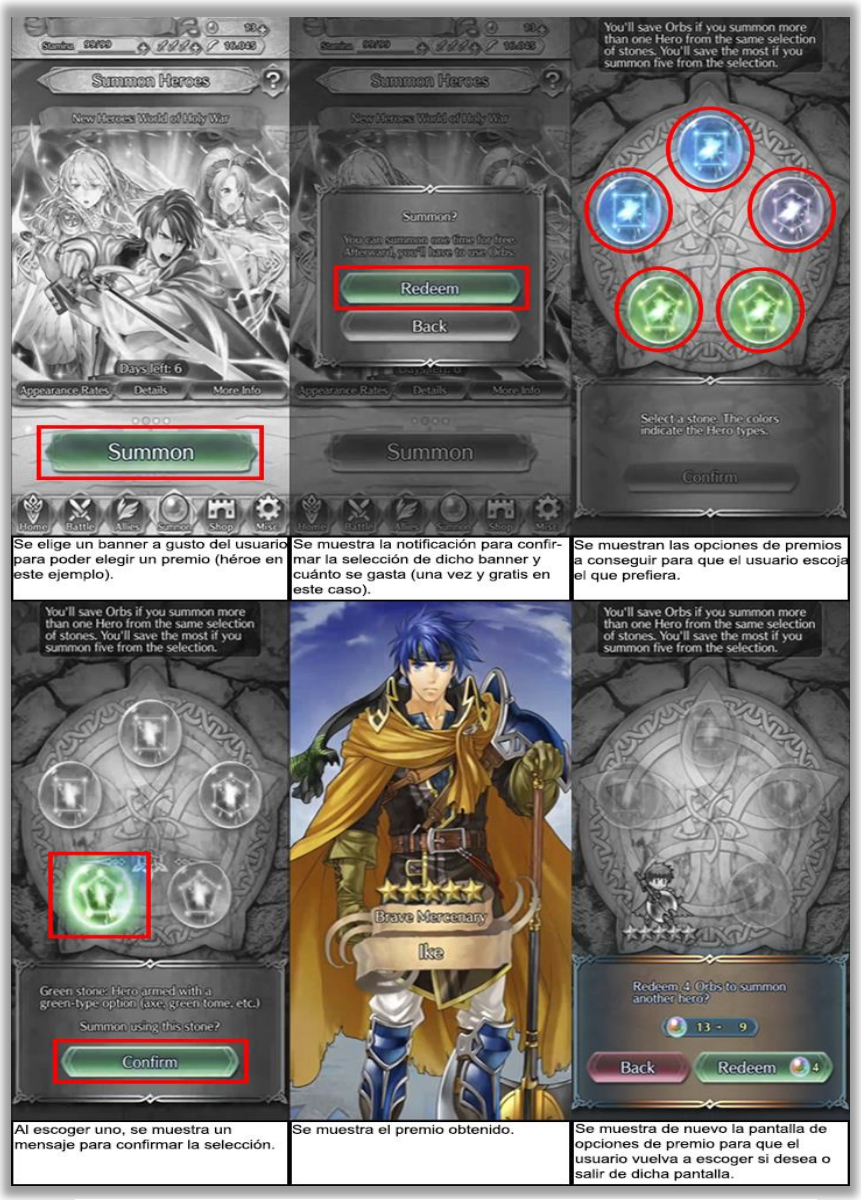
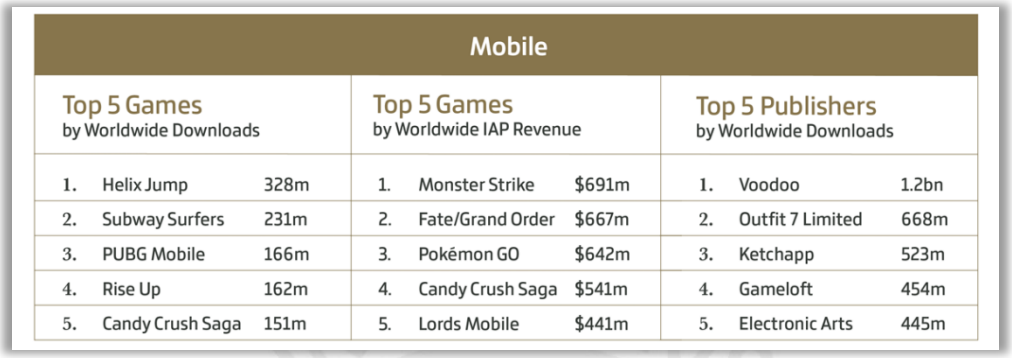


Figura 2. Ejemplo de un *rolleo* de Gacha (2019).
Fuente: Fire Emblem Heroes (Captura propia)

Una característica de estos juegos es que cuentan con compras dentro de la aplicación (*in-app purchase* o IAP) o micro transacciones. Como explica la página de soporte de Apple, existen 3 tipos de IAP, los de suscripción, de consumo y no consumo. Dependiendo de la situación, el jugador puede optar por uno de estos tipos de compra y obtener beneficios dentro del juego. Esto se puede apreciar en el juego *Fire Emblem Heroes* (FEH por sus siglas), el cual tiene como sistema monetario dentro del juego (*in- game currency)* orbes (*orbs*). Según indica

la página de soporte del juego, los orbes tienen múltiples funciones. Por ejemplo, conseguir personajes o restaurar la energía del jugador para poder seguir usando la aplicación. Si se desea conseguir más de estas "monedas", se tiene la opción de comprarlas con dinero real en las tiendas virtuales de Apple o Android (Fire Emblem Heroes, 2019).



Mobile

Top 5 Games by Worldwide Downloads			Top 5 Games by Worldwide IAP Revenue			Top 5 Publishers by Worldwide Downloads		
1.	Helix Jump	328m	1.	Monster Strike	$691m	1.	Voodoo	1.2bn
2.	Subway Surfers	231m	2.	Fate/Grand Order	$667m	2.	Outfit 7 Limited	668m
3.	PUBG Mobile	166m	3.	Pokémon GO	$642m	3.	Ketchapp	523m
4.	Rise Up	162m	4.	Candy Crush Saga	$541m	4.	Gameloft	454m
5.	Candy Crush Saga	151m	5.	Lords Mobile	$441m	5.	Electronic Arts	445m

Figura 3. Ranking de juegos móviles (2018).
Fuente: GamesIndustry

Los juegos del tipo *Gacha* son accesibles para cualquier usuario de Smartphone o Tablet. En el año 2018, lograron posicionarse entre los videojuegos más vendidos a nivel mundial con *Monster Strike* que tuvo una recaudación total de 691 millones de dólares y *Fate/Grand Order* con una recaudación total de 667 millones de dólares (Batchelor, 2018) (ver Figura 3). Una de las formas de obtener estas ganancias, es ofrecer promociones a los jugadores para motivarles a invertir dentro del juego. A medida que el jugador invierte, la compañía desarrolladora genera más ingresos, logrando así una estabilidad entre la satisfacción del jugador y las ganancias de la empresa. Este es el caso de la compañía miHoYo con su juego *Honkai Impact 3rd* el cual logró obtener el puesto número uno en el ranking de la *China App Store* en Agosto del 2018. Para conseguir un personaje (*valkyrie*) se tiene que invertir en alguno de los *Gachas*, uno de poca inversión de cristales (*Crystals*) u otro diez veces el valor del primero. Para poder obtener una valkiria de alto rango, se les propone a los usuarios gastar cristales en el *Gacha* que cuesta más, asegurando una mayor probabilidad de obtención. Por otra parte, si no tienen suficientes recursos para invertir en esa promoción, se les da la opción de gastar en el *Gacha* de bajo costo

asegurando que, si no han obtenido una valkiria de alto rango en nueve intentos, en el décimo intento obtendrán una de alto rango (Honkai Impact 3rd, 2019).

Como explica Baglin (2017) los juegos *Gacha* son como un juego en donde se permite a los usuarios la opción de conseguir personajes u objetos. Dependiendo del juego, pueden elegir entre los distintos tipos de *Gachas* para gastar/tirar/"rollear" (término coloquial proveniente del inglés *roll*). Un ejemplo claro es el juego *Fate/Grand Order* (FGO por sus siglas) el cual tiene 3 tipos. Uno de eventos exclusivos (*Event Gacha*) donde se pueden obtener personajes (*Servants*) y cartas de equipamiento (*Craft Essence*) limitados por un tiempo determinado; otro de historia (*Story Gacha*) en el cual, a diferencia del exclusivo, está disponible siempre y contiene a la mayoría de personajes, exceptuando a los limitados durante los eventos; y por último el de amigos (*Friend Gacha*), donde se obtienen personajes y cartas de equipo básicos.

Sin embargo, los juegos *Gacha* no siempre llegan a ser los mejores debido a un factor clave, el cual sería su propia mecánica. Para poder obtener algún recurso especial del juego, se debe de gastar, ya sea con el sistema monetario del juego o con dinero real, pero esto no asegura que se conseguirá lo que uno desea. Al cumplir con la transacción, el juego te da a cambio lo que puedas obtener bajo las especificaciones generadas por el algoritmo del juego. Este resultado puede ser satisfactorio para el jugador, como también lo contrario. En el caso donde el jugador no obtuvo lo que quería, tiene que volver a invertir, esto puede llegar a repetirse varias veces hasta que ya no tenga recursos o ya no le interese seguir jugando. El tener que invertir dinero real en el juego, puede generar disgusto en los jugadores y en consecuencia dejar de utilizar la aplicación (Baglin, 2017).

Este efecto produce una serie de emociones que experimentan los usuarios al momento de obtener un premio. Según la Teoría de la Experiencia Óptima (Teoría del Flow), el usuario que está pendiente de una actividad que le fascina y cumple con algunos de estos elementos

(Cowley, Charles, Black & Hickey, 2008), disfrutará la actividad que está realizando y querrá seguir con ésa emoción; por otro lado, de no llegar a cumplirse, la emoción que se produce puede direccionarse a la ansiedad de no poder obtener el premio que desee, o al aburrimiento de seguir intentando y no conseguirlo. Como resultado, las empresas responsables de los juegos pierden a sus jugadores y, en consecuencia, también ganancias. Una medida alternativa es modificar el algoritmo encargado del sistema de premiación. En el artículo de Nakamura, se hace mención de que las compañías japonesas el solo dedicarse a incrementar los elementos exclusivos que ofrece el *Gacha*, ha generado que sus juegos pierden el nivel de innovación que en un principio los hizo relevantes, siendo superado por otro tipo de juegos de compañías chinas (Nakamura, 2018). Al pertenecer estos usuarios a una comunidad competitiva, considerarán cualquier ventaja que exista sobre otro jugador y con ello, el impulso y motivación de invertir en el juego.

La mayoría de juegos *Gacha* informan al jugador el porcentaje de probabilidad de obtener algún objeto o personaje, mientras más importante sea dicho elemento, menor es la probabilidad de obtenerlo. En el ejemplo de FEH cuando se presenta un nuevo personaje especial, muestran el ratio de aparición del mismo (Nintendo, 2019). Al modificar dicho algoritmo, se podría generar una mayor satisfacción a los usuarios, pero para las empresas significaría que los jugadores dejarían de invertir dinero, provocando que los ingresos de la empresa disminuyan. Si se realiza otra modificación donde se priorice la inversión sobre la satisfacción, esto puede conllevar a la premisa inicial donde los jugadores dejen de consumir la aplicación.

Así mismo, las empresas responsables han estado bajo observación y sujetas a medidas legislativas. Esto se debe a incidentes donde las demandas a estos juegos llegaron a un punto donde se involucraban a usuarios menores de edad gastando grandes sumas de dinero, logrando aumentar la intensidad de las normas y leyes para el consumo de estos juegos; además de la

prohibición de uno de los tipos de *Gacha* en Japón en el año 2012 (De Vere, 2012). En efecto, este hecho se tomó como referencia para que las compañías consideren estas nuevas reglas para la creación de juegos *Gacha*. Sin embargo, las constantes investigaciones regulatorias por parte de entidades políticas alrededor del globo hacen que la dificultad de mantener un equilibrio aún persista (McKenzie, 2018).

Por tal motivo, existe el problema de lograr calcular el equilibrio entre satisfacción del cliente y maximizar las ganancias de la empresa. Para este proyecto se optó por un tipo algoritmo metaheurístico por ser el más adecuado para encontrar una solución satisfactoria ante problemas de optimización combinatoria. Para su desarrollo se enfoca en dos características, la intensificación, que busca las soluciones entorno a una solución principal dada; y la diversificación, que explora zonas no visitadas para la búsqueda de soluciones óptimas. El caso más común de la aplicación de este tipo de algoritmo está en el Problema del Viajante (*Travelling Salesman Problem* o *TSP*) que busca la mejor ruta para vender sus productos visitando una sola vez cada ciudad (Muhammad, Gao, Qaisar, Abdul, Muhammad, Usman, Aleena & Shadid, 2018). Primero se escoge una ruta al azar y a partir de esa solución inicial, se compara con otras soluciones que pueden dar un mejor resultado.

Dentro de este grupo, el algoritmo propuesto será el *Cuckoo Search Algorithm* por tener aplicación en problemas de análisis de estabilidad y menor tiempo computacional. (Mohammad, Zain & Nazira, 2014). Al contar con dos capacidades de búsqueda, una local y una global, permite que el espacio de búsqueda sea explorado más eficientemente. Además, obtiene mayores ventajas sobre otros algoritmos al poder combinar los dos tipos de búsqueda bajo estándares Gaussianas. Asimismo, ha tenido diferentes aplicaciones, por ejemplo, se ha logrado utilizar para resolver problemas de programación de enfermería, para el diseño de marcos de acero e incluso para resolver cálculos termodinámicos (Yang & Deb, 2014). Este algoritmo se implementará con el fin de brindar una buena solución ante la cuestión planteada.

Para verificar el resultado de esta investigación, se realizará una comparación con un simulador de este tipo de juegos. Con todo esto, este proyecto de fin de carrera consistirá en aplicar un algoritmo metaheurístico con el propósito de encontrar la estabilidad más adecuada en estos tipos de juegos.

1.2 Objetivos

1.2.1 Objetivo general

Implementar un algoritmo metaheurístico *cuckoo search*, el proceso de roleo en videojuegos con sistema de premios en conjunto.

1.2.2 Objetivos específicos

O 1. Definir una función objetivo que permita estimar el balance entre el peso del premio obtenido y el beneficio de la empresa.

O 2. Adaptar el algoritmo metaheurístico *cuckoo search* propuesto al contexto especificado.

O 3. Implementar un simulador y un algoritmo metaheurístico *cuckoo search* en un lenguaje de programación que permitan representar la acción del juego.

O 4. Diseñar un prototipo de juego para probar el funcionamiento del algoritmo.

O 5. Realizar una experimentación numérica para comparar el desempeño del algoritmo metaheurístico y el simulador.

1.2.3 Resultados esperados

Para el primer objetivo específico (O1):

R 1. Función objetivo basada en el beneficio de la empresa y el peso del premio obtenido que se usará en el algoritmo *cuckoo search*.

Para el segundo objetivo específico (O2):

R 2. Estructura de datos del algoritmo *cuckoo search*.

R 3. Pseudocódigo del algoritmo *cuckoo search*.

Para el tercer objetivo específico (O3):

R 4. Módulo del algoritmo *cuckoo search* implementado.

R 5. Módulo del simulador implementado.

Para el cuarto objetivo específico (O4):

R 6. Prototipo de videojuego funcional que utilice el algoritmo y el simulador.

Para el quinto objetivo específico (O5):

R 7. Informe de experimentación numérica con las pruebas y resultados realizados.

1.2.4 Mapeo de objetivos, resultados y verificación

Tabla 1: Objetivos, Resultados y Medios de Verificación
Fuente: Elaboración propia.

Objetivo: Definir una función objetivo que permita estimar el balance entre el peso del premio obtenido y el beneficio de la empresa.		
Resultado	**Meta física**	**Medio de verificación**
Función objetivo que se usará en el algoritmo *cuckoo search*.	Documento con la función planteada.	Revisión por especialista.
Objetivo: Adaptar el algoritmo metaheurístico *cuckoo search* propuesto al contexto especificado.		
Resultado	**Meta física**	**Medio de verificación**
Estructura de datos del algoritmo metaheurístico.	Documento con conjunto de datos.	Prueba y seguimiento con un conjunto de datos y resultados.
Pseudocódigo del algoritmo *cuckoo search*.	Documento con pseudocódigo.	Prueba y seguimiento con un conjunto de datos y resultados.
		Revisión por especialista.
Objetivo: Implementar un simulador y un algoritmo metaheurístico *cuckoo search* en un lenguaje de programación que permitan representar la acción del juego.		
Resultado	**Meta física**	**Medio de verificación**
Módulo del algoritmo *cuckoo search* implementado.	Software del algoritmo *cuckoo search*.	Pruebas unitarias.
Módulo del simulador.	Software del simulador.	Pruebas unitarias.

Objetivo: Diseñar un prototipo de juego para probar el funcionamiento del algoritmo.		
Resultado	**Meta física**	**Medio de verificación**
Prototipo de videojuego funcional que utilice el algoritmo y el simulador.	Software	Pruebas unitarias.
		Pruebas funcionales.
		Documento de diseño de juego (*Game Design Document*)
		Revisión por especialista.
Objetivo: Realizar una experimentación numérica para comparar el desempeño del algoritmo metaheurístico y el simulador.		
Resultado	**Meta física**	**Medio de verificación**
Informe de experimentación numérica con las pruebas y resultados realizados.	Documento de experimentación numérica.	Experimentación numérica y discusión de resultados.

1.3 Herramientas y Métodos

En la siguiente tabla (ver Tabla 2) se mostrarán las herramientas, métodos y metodologías que se usarán en el presente proyecto de fin de carrera.

Tabla 2: Herramientas, Métodos y Metodologías a usar
Fuente: Elaboración propia.

Resultados esperados		**Herramientas, métodos y metodologías**
R1	Función que se usará en el algoritmo metaheurístico y en el algoritmo competidor.	
R2	Estructura de datos del algoritmo metaheurístico.	
R3	Pseudocódigo del algoritmo *cuckoo search*.	PSeInt
R4	Módulo del algoritmo *cuckoo search* implementado.	Microsoft Visual Studio
		C#
R5	Módulo del algoritmo competidor implementado.	Microsoft Visual Studio
		C#
R6	Videojuego que muestre el funcionamiento de los algoritmos.	Unity
		Android Studio
R7	Informe de experimentación numérica con las pruebas y resultados realizados.	Comparación de dos muestras: • Prueba Z

		• Prueba de Kolmogorov-Smirnov • Prueba F de Fisher
		Microsoft Excel

1.3.1 Herramientas

1.3.1.1 PSeInt

PSeInt es una herramienta que permite la programación y ejecución de algoritmos en pseudolenguaje. Este software presenta conjunto de ayudas y asistencias para ayudar al programador en encontrar errores y entender la lógica de los algoritmos (PSeInt, 2003).

Se ha decidido utilizar esta herramienta porque permite programar en pseudocódigo el algoritmo propuesto, además de brindar un diagrama de flujo para mostrar su funcionamiento y también poder editarlo.

1.3.1.2 Visual Studio IDE

Visual Studio es un entorno de desarrollo integrado (*IDE*), el cual sirve para el desarrollo de aplicaciones desde Servicios Web XML como también ASP. Es compatible con múltiples lenguajes de programación como por ejemplo C++, C#, Visual Basic, Java, Python, entre otros con funciones .NET Framework. Este IDE está habilitado tanto en Windows como también en MAC. Visual Studio permite a los desarrolladores compilar y crear juegos, aplicaciones y servicios web en cualquier lenguaje (Visual Studio, 2019).

Se ha decidido utilizar este IDE porque facilita el desarrollo de proyectos de programación. A parte, cuenta con funcionalidades que ayudan a simplificar la edición del código, tales como autocompletar el código, refactorizar código, limpieza de código y alerta de errores. Esta herramienta se utilizará para implementar los algoritmos.

1.3.1.3 C#

Es un lenguaje de programación orientado a objetos que permite a los desarrolladores crear una gran diversidad de aplicaciones que se ejecutan en .NET. C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común al ser sencilla y fácil de aprender. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes (MICROSOFT, 2015).

Se ha decidido utilizar este lenguaje ya que es compatible con Visual Studio al ser pertenecientes de la misma compañía. Asimismo, otro motivo de elección fue por contar con el conocimiento previo de este lenguaje.

1.3.1.4 Unity Personal

Unity es un motor de desarrollo completamente integrado para la creación de contenido 3D interactivo. Proporciona una funcionalidad completa y lista para usar para ensamblar contenido de alta calidad y alto rendimiento y publicar en múltiples plataformas, como en consolas de videojuegos, smartphones y tablets. Unity ayuda a desarrolladores y diseñadores independientes, pequeños y grandes estudios, corporaciones multinacionales, estudiantes y aficionados a reducir drásticamente el tiempo, el esfuerzo y el costo de hacer juegos (Unity, 2019).

Se eligió esta herramienta por ser una de las más conocidas en desarrollo de juegos y tener accesibilidad en dispositivos móviles. Se utilizará esta herramienta para la creación de un juego simple donde se usará los algoritmos y ver su funcionamiento de forma visual.

1.3.1.5 Android Studio

Android Studio es un IDE oficial para el desarrollo de aplicaciones en los distintos dispositivos de Android. Basado en el software IntelliJ IDEA, integra un entorno para software

e incorpora herramientas de desarrollo y edición de código. También, Android Studio cuenta con su propio emulador, modelos de código y permite integración con GitHub. Este IDE está habilitado para Windows, MAC y Linux (Android Studio, 2017).

Esta herramienta fue escogida para complementar con el desarrollo del juego móvil. Se utilizará esta herramienta para importar las librerías que se necesitarán en la plataforma de Unity.

1.3.2 Métodos y procedimientos

1.3.2.1 Comparación de dos muestras

Método que permite comparar dos muestras, las cuales son resultados de la ejecución de una operación específica realizada por distintos elementos. Se determina cuál de los dos elementos es el que genera una muestra más significativa para dicha operación (Box, Hunter, & Hunter, 1993).

Se realizará esta comparación de dos tratamientos para determinar si el algoritmo propuesto genera una solución más óptima que el algoritmo competidor y contra el simulador. Para llevar a cabo este método se realizarán las siguientes pruebas:

- **Prueba Z:** Prueba estadística utilizada para determinar si la media de dos poblaciones es diferente cuando las varianzas son conocidas y el tamaño de la muestra es lo suficientemente grande. Se asume que la prueba tiene una distribución normal y que la desviación estándar es conocida, de tal modo que se pueda llevar a cabo una prueba Z exacta (Lin & Mudholkar, 1980).
- **Prueba Kolmogorov-Smirnov:** Prueba permite medir el grado de concordancia que existe entre la distribución de un conjunto de datos y una distribución específica. Su objetivo es determinar si los datos provienen de una población que tiene una supuesta distribución determinada (García, Gonzáles & Jornet, 2010).

- **Prueba F de Fishe**r: Prueba estadística que sirve para comparar varianzas mediante una distribución de probabilidad continua F. Se utiliza cuando se busca una comparación entre poblaciones para determinar cuál tiene una mayor variación sobre otra (Ycart, Pont & Fournié, 2014).

Capítulo 2. Marco Conceptual

En este apartado se explican los conceptos relacionado a la problemática, como también, conceptos básicos de algoritmos y heurística.

2.1 Conceptos Generales

2.1.1 Aplicación Móvil

Una aplicación móvil es una aplicación informática diseñada para ser consumida en teléfonos inteligentes (*smarthpones*), tablets (*Tablet*) y otros dispositivos móviles. Permiten al usuario efectuar un conjunto de tareas de cualquier tipo facilitando las actividades a desarrollar. (Servisoftcorp, 2019).

2.1.2 Videojuego

Consultando la Real Academia Española (RAE), define al videojuego como un dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor, una computadora u otro dispositivo electrónico (Real Academia Española [RAE], 2019). A parte, los videojuegos, además de ser un entretenimiento operativo, pueden ser complejos y ambiguos. También, provee de información y rutinas de procesos educativos en la medida en que el diseño, la estética y operatividad de la propuesta lúdica obliga a transitar al jugador / operador por una historia narrada en la que éste interviene, alcanzando habilidades para superar los retos planteados (Gómez, 2018).

2.1.3 Videojuegos gratuitos (*Free to Play*)

Los videojuegos gratuitos, como dice el nombre, son juegos de acceso gratuito de su contenido para los jugadores. Estos juegos tienen varios tipos, los más representativos son los *sharewares*, los cuales son una prueba para convencer a los usuarios de comprar el juego completo, también conocidos como demos. Y *freemium*, los cuales ofrecen una versión

completa del producto, pero se les hace un cargo de micro transacciones para acceder a funciones *premium* y objetos virtuales (*virtual good*) (Tack, 2013).

2.1.4 Simulador

Según la Real Academia Española (RAE), describe a un simulador como un aparato que reproduce el comportamiento de un sistema en determinadas condiciones, aplicado generalmente para el entrenamiento de quienes deben manejar dicho sistema (Real Academia Española [RAE], 2019). Existen distintos tipos de simuladores para diferentes áreas como de transporte, negocio, música y entretenimiento.

2.1.5 Juegos *Gacha*

Los juegos *Gacha* son videojuegos que simulan a la máquina de gashapon. El gashapon es una máquina dispensadora de juguetes de cápsula populares en Japón. La palabra gashapon proviene de la onomatopeia de los dos sonidos que produce la máquina "*gasha"*, por el sonido al utilizar la manija de la máquina, y "*pon"* por el sonido que produce la cápsula al caer en la bandeja de recojo.

Figura 4. Ejemplo de un cartel informativo (Banner) del Gacha de un juego (2019).
Fuente: Fate/Grand Order (Captura propia)

La gran mayoría de estos juegos son videojuegos gratuitos (F2P) para móviles donde los jugadores gastan monedas virtuales en estas máquinas virtuales. Usualmente en estos juegos se encuentran personajes, cartas u otros objetos que el jugador puede obtener mediante el *Gacha* (Famularo, 2017).

2.1.6 Compras dentro de la aplicación (*in-app purchase*)

Las compras desde la aplicación son contenidos o suscripciones adicionales que puede comprar en aplicaciones en el dispositivo o computador (Apple, 2019).

2.1.7 Moneda del juego (*in-game currency*)

La moneda de juego es un término que describe todas las monedas virtuales utilizadas en los juegos en línea. Dichas monedas se pueden usar para comprar artículos dentro de un juego para mejorar o actualizar el juego, así como el usuario. La moneda de juego se puede obtener de dos maneras. Una es cambiar otra moneda, generalmente nacional, a una moneda de juego, mientras que la segunda opción es ganar la moneda a través de la actividad en el juego (Community Currency Knowledge Gateway, 2014).

2.2 Conceptos generales de algoritmia

2.2.1 Algoritmo

Es un conjunto de reglas o instrucciones para efectuar algún cálculo, usado frecuentemente en una máquina. No debe ser subjetivo. En base a un estado inicial, sigue los pasos establecidos para llegar a un estado final y brindar una solución (Brassard & Bratley, 1998).

2.2.2 Heurística

Según la Real Academia Española (RAE), se define como:

- Técnica de indagación y descubrimiento.
- Búsqueda o investigación de documentos o fuentes históricas.

- En algunas ciencias, manera de buscar la solución de un problema mediante métodos por tanteo, reglas empíricas, etc. (RAE, 2019).

En usabilidad, una evaluación heurística tiene como objetivo medir la calidad de cualquier sistema interactivo en relación a su sencillez de aprendizaje y usado para un determinado público en un contexto dado (González, Lorés & Pascual, 2006).

En el área de programación, un algoritmo heurístico consiste en encontrar una solución mediante pruebas no necesariamente directas, generando candidatos de las posibles soluciones. Estos candidatos son sometidos a experimentos siguiendo un criterio, si no es el esperado, se rechaza y se vuelve a repetir el proceso.

2.2.3 Metaheurística

El término metaheurístico fue introducido por Glover (1986) al definir una clase de algoritmos de aproximación que combinan heurísticos tradicionales con estrategias eficientes de exploración del espacio de búsqueda (Blum & Roli, 2003).

Osman y Kelly (1996) definen la metaheurística como métodos de aproximación diseñados para resolver problemas de optimización combinatoria, en los que las heurísticas comunes no son efectivas. Además, proporcionan un marco general para crear nuevos algoritmos híbridos, logrando combinar diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

Una de las ventajas de estos algoritmos es su flexibilidad, esto se debe al hecho que pueden abordar un amplio plantel de problemas y también, poder combinarlos con otros algoritmos. A parte, tienen dos características principales para su desarrollo, la intensificación, ya que buscan soluciones entorno a una solución principal seleccionada; y la diversificación, porque permite explorar zonas de soluciones no frecuentadas para la búsqueda de soluciones (Vélez & Montoya, 2007).

2.2.4 *Cuckoo Search*

Es una búsqueda metaheurística inspirada en el cuco común (*cucko*), esta ave utiliza una estrategia conocida parasitismo de puesta. Esta estrategia consiste en usar otras especies para que cuiden y críen su descendencia. En el caso del cuco, la hembra pone sus huevos en otros nidos de aves mientras éstas están ausentes. Para no ser descubiertas, los huevos tienen patrones que se asemejan a los del nido puestos.

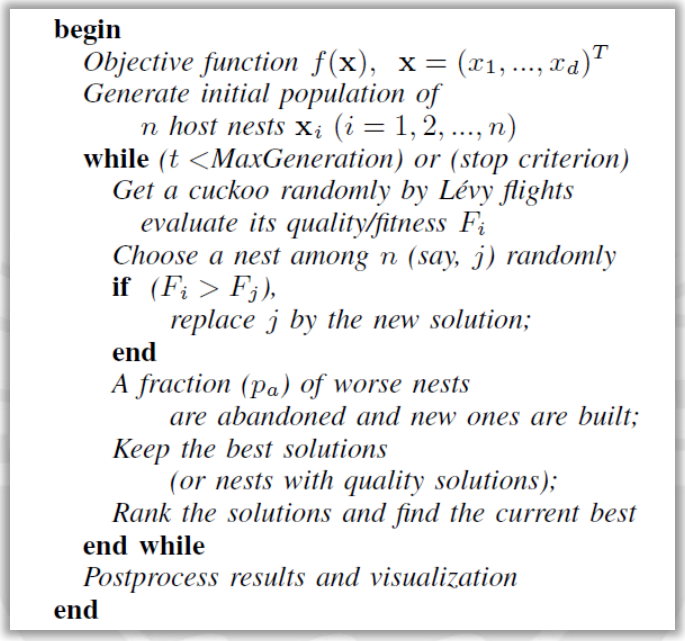


Figura 5. Pseudocódigo de Cuckoo Search (2009).
Fuente: Cuckoo Search via Lévi flights

En base a la Figura 5, la búsqueda cuco se rige de tres ideas principales (Yang & Deb, 2009):

- Cada cuco pone un huevo a la vez y deja ese huevo aleatoriamente en un nido escogido.
- Los mejores nidos con huevos de alta calidad se trasladarán a las siguientes generaciones.
- El número de nidos anfitriones disponibles es fijo, y el ave huésped descubre el huevo puesto por un cuco con una probabilidad *p* (0 o 1).

Caso se llegue a encontrar el huevo, el ave madre puede optar por dos opciones, tirar el huevo o abandonar el nido y construir uno completamente nuevo. En resumen, cada huevo representa una solución y cada huevo cuco en el nido representa una nueva solución. El objetivo es usar la potencial nueva solución, el huevo cuco, para remplazar a las no tan buenas soluciones del nido (Yang & Deb, 2009).

Capítulo 3. Estado del Arte

3.1 Objetivo

Esta sección tiene como objetivo presentar alternativas de solución para el problema descrito de encontrar un equilibrio entre satisfacción del cliente y las ganancias de las empresas desarrolladoras de juegos con sistemas de premiación. Primero se revisará las soluciones encontradas y finalmente se dará una conclusión de todo lo explicado.

3.2 Método usado en la revisión del estado del arte

Para la revisión de artículos, investigaciones, tesis y conferencias se utilizó el método de revisión sistemática. El proceso de investigación se realizó en las bases de datos de sistema de biblioteca de Tesis de la Facultad de Ciencias e Ingeniería de la PUCP, Skopus, ACM, IEE Xplorer y ScienceDirect. Sin embargo, se encontraron una cantidad muy limitada de soluciones, por tal motivo se amplió la búsqueda en la base de datos de Google Académico (*Google Scholar*) el cual guiaba a otras bases de datos de repositorios de investigación.

Por el método PICOC, podemos identificar los elementos correspondientes:

Tabla 3: Análisis PICOC
Fuente: Elaboración propia.

Población	Intervención	Comparación	Salida (*Outcome*)	Contexto
Algoritmos de juegos tipo *Gacha*	Algoritmo metaheurístico *Cuckoo Search*	Algoritmo metaheurístico y familia del algoritmo de juegos *Gacha* (simulador)	Métodos para la estimación de nivel de satisfacción del usuario	Juegos móviles

La lista de palabras utilizadas para la investigación fue: *mobile, game(s),* juegos, *gacha, loot box, metaheuristic,* metaheurístico, *video game(s),* videojuegos, *comparison, between, design, solutions, problem, algorithm(s),* algoritmo(s), *cuckoo search.*

En una primera instancia se identificaron las siguientes preguntas a resolver:

- ¿Qué alternativas se han tomado para lograr la satisfacción del jugador en juegos *Gacha*?
- ¿Cuáles algoritmos se han utilizado para el desarrollo de juegos *Gacha*?
- ¿Qué otros algoritmos metaheurísticos se han comparado con el algoritmo de un juego *Gacha*?

Al realizar las búsquedas en las diferentes bases de datos, mostraban a lo mucho, tres resultados por buscador. Además, el contenido de los artículos no llegaba a tener relación con el problema, pero de todas formas se consideraron para investigar las referencias de esos artículos. Seguidamente, se optó en ampliar la búsqueda con las siguientes preguntas:

- ¿Qué videojuegos utilizan algoritmos metaheurísticos?
- ¿Qué otros algoritmos metaheurísticos se han comparado con *cuckoo search*?

El resultado logró ampliar un poco más los resultados, en especial la pregunta referente a la comparación de algoritmos y cuáles se han utilizado. De esta cantidad se consideró artículos que complementaban en la definición de conceptos y documentos que se aproximarán al contexto de investigación. Además, se utilizaron criterios para mejorar la selección y filtrado de lo encontrado (ver Tabla 5).

La cadena de búsqueda utilizada fue:

Tabla 4: Cadena de Búsqueda
Fuente: Elaboración propia.

1	mobile game OR juegos móviles AND (gacha OR loot box AND (algorithm OR algoritmo AND (problem OR solution)))
2	design AND mobile game AND (gacha OR loot box)
3	comparison AND between AND cuckoo search
4	video game OR videojuegos AND (metaheuristic algorithm OR algoritmo metaheuristico)
5	video game OR videojuegos AND (cuckoo search)

Los criterios de inclusión y exclusión utilizados fueron:

Tabla 5: Criterios de Inclusión y Exclusión
Fuente: Elaboración propia.

Criterios de inclusión		Criterios de exclusión	
1	Documentación referente a juegos *Gacha* desde creación, investigación y análisis.	1	Documentación con idioma distinto al español o inglés.
2	Documentación referente al desarrollo de videojuegos utilizando algoritmos metaheurísticos.	2	Documentación publicada con antigüedad mayor a 19 años.

Los criterios de inclusión logran complementar a las preguntas referentes a la utilización del algoritmo, ampliando la búsqueda inicial. El segundo criterio de exclusión se utiliza para considerar la información que tenga impacto o uso reciente.

En total se logró conseguir una cantidad de 80 elementos entre artículos, documentos de investigación y tesis. Aplicando los criterios previamente descritos, se redujo la cantidad inicial, logrando obtener 45 artículos para su revisión y uso (ver Tabla 6).

Tabla 6: Cantidad de Documentos Clasificados
Fuente: Elaboración propia.

Clasificación	Cantidad
Documentos encontrados con las cadenas de búsqueda	45
Documentos seleccionados bajo criterios de inclusión	20
Documentos no considerados bajo criterios de exclusión	15
Documentación seleccionados finalmente	**5**

Tabla 7: Cantidad de Documentos por Base de Datos
Fuente: Elaboración propia.

Bases de datos	Cantidad
Repositorio digital de tesis PUCP	2
Scopus	0
ACM	1
IEEE Xplore	2
ScienceDirect	5
Google Scholar	35

3.3 Revisión y discusión

3.3.1 Soluciones para la creación de un *Gacha*

Primero se revisará una solución con respecto al diseño de un sistema *Gacha*.

3.3.1.1 Cómo diseñar un sistema *Gacha*

En esta presentación, define tres aspectos importantes que son fundamentales para la creación un sistema *Gacha* funcional: profundidad, amplitud y deseo. Por cada aspecto hace una explicación indicando las características de cada una, las sugerencias para modelar el sistema y como estos se aplican en otros juegos. A la vez, hace mención que, si la empresa no tiene bien definidos estos aspectos claves, su juego no perdurará en el tiempo, perderá usuarios y, en consecuencia, pérdida de ganancias y la eliminación del juego (Telfer, 2017).

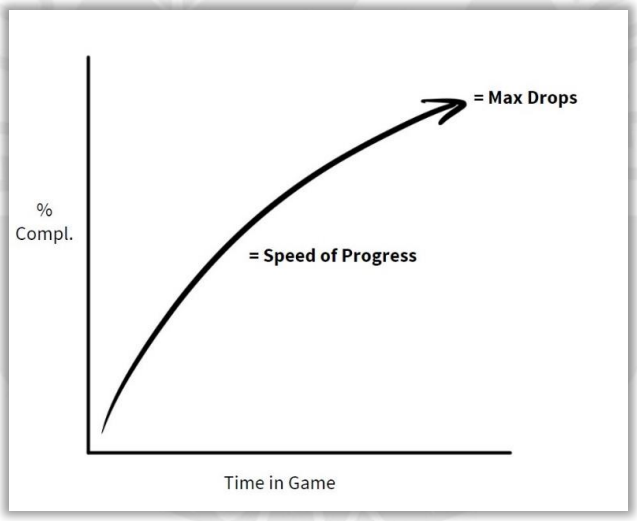


Figura 6. Gráfica de cómo debería de sentirse el Gacha (2017).
Fuente: Pocket Gamer Helsinki 2017: recipe for Strong Gacha

3.3.2 Soluciones para predecir el consumo en juegos *Free To Play*

Al seguir revisando las soluciones exploradas, se hallaron alternativas donde se evalúa al jugador y las posibilidades que éste logre gastar dinero en estos juegos.

3.3.2.1 Predecir Decisiones de Compra en Juegos Móviles *Free-to-Play*

En este trabajo se implementa algoritmos de *Machine Learning* para predecir dentro de un conjunto de datos de jugadores, cuánta cantidad aproximada llegan a hacer compras dentro

del juego. Los autores justifican el uso de este tipo de algoritmos ya que eran más rápidos de implementar y accesibles a comparación de los modelos *Hidden Markov Models* o *Neural Network*. Para la clasificación de los datos los miembros responsables optaron por dos modelos: un modelo de clasificación enfocado en la predicción de si se realizará una compra y un modelo de regresión enfocado en la cantidad de compras que realizará el usuario.

Para el modelo de clasificación, generan jugadores que van a pagar (*premium*) bajo la técnica *Synthetic Minority Over-sampling Technique-Nominal Continuous* (SMOTE-NC). Según los autores, SMOTE-NC genera data para popular el conjunto de datos bajo las condiciones de proporción del número de la minoría contra la mayoría. Para el modelo de regresión proponen el uso de *Poisson Regression Tree* (PRT). Al igual que la justificación inicial, este árbol de regresión son más fáciles de interpretar y aprender.

Para los dos modelos, se logra obtener resultados que, al combinarlos, se logra apreciar una distinción entre usuarios y sus comportamientos esperados (Sifa, Hadiji, Runge, Drachen, Kersting & Bauckhage, 2015).

3.3.2.2 *Churn Prediction* para Jugadores de Alto Valor en Juegos Sociales Casuales

En este artículo se busca predecir el *Churn* de jugadores potenciales en este tipo de juegos utilizando algoritmos *Machine Learning*. El objetivo planteado de los autores es lograr predecir cuántos jugadores potenciales haya el riesgo que abandonen el juego.

Para la resolución, concretan el problema lo que implica la definición de su población (jugadores potenciales), su base de evaluación (número de días consecutivos sin jugar) y su "jugador churn" quien será su dato de comparación. Para la creación del modelo los autores tienen un conjunto de datos donde se aplican cuatro algoritmos de búsqueda donde se evalúa su desempeño con respecto a la predicción *churn* obtenido anteriormente. Seguido se combina uno de estos algoritmos con un modelo *Hidden Markov Models*, con esta comparación se

estima un aproximado con jugadores que presenten extraños comportamientos. Finalmente, se hace una experimentación numérica para determinar si la evaluación inicial llega a igualar o aproximarse a la comparación final.

Al final del artículo concluyen que esta predicción ayuda a las empresas a evaluar, en base a los resultados obtenidos, si necesita realizar cambios o agregar nuevas mecánicas en sus juegos para no perder sus jugadores potenciales. (Runge, Gao, Garcin & Faltings, 2014).

3.3.3 Soluciones para crear videojuegos con algoritmos metaheurísticos

Extendiendo el área de búsqueda, se halló documentación relacionado a la creación de videojuegos utilizando algoritmos metaheurísticos.

3.3.3.1 *AntBot*: Colonización de Hormigas para Videojuegos

Este apartado tiene como fin la implementación de *bots*/*non player characters* (*NPC*) bajo un marco de trabajo enfocado en la inteligencia de enjambre, utilizando el algoritmo de la colonia de hormigas (*Ant Colony Optimization*).

Para el desarrollo del *bot*, definieron tres partes, la primera donde se utiliza el algoritmo para realizar una acción en base a la recolección de información del estado actual del juego. Este algoritmo es aplicado en el juego *Ms. Pac-Man*, para lograr que los fantasmas del juego tengan mayor diversidad ante la respuesta del jugador. La segunda parte es el algoritmo de la selección de la acción, cuando el jugador avanza se mide la distancia entre este y los fantasmas y se selecciona el mejor camino de hormigas. Para el papel de los fantasmas, la situación más peligrosa para el jugador. Finalmente, extienden el algoritmo para que pueda seleccionar la mejor acción que deben tomar los cuatro fantasmas para el estado actual del juego, además de tener en cuenta que, si están en estado de ser comidos, alejarse del jugador.

Los autores concluyen la colonia de hormigas tiene potencial en generar *bots* para juegos de acción real. También, resaltan que han brindado un nuevo marco de trabajo en inteligencia

de enjambre al ver un corto estudio de este algoritmo en este tipo de problemas. (Recio, Martin, Estébanez & Saez, 2012).

3.3.3.2 Videojuego RPG (*Role-Playing-Game*) basado en algoritmos evolutivos

El autor de este proyecto tiene como propósito utilizar algoritmos evolutivos en la jugabilidad del videojuego, específicamente el uso de algoritmos genéticos. El autor detalla que la computación evolutiva ayuda en el desarrollo; realizar el proceso de aprendizaje en caso de la inteligencia artificial y elaboración de mapas y mecánicas en el contenido del videojuego. Para el diseño del algoritmo genético, el autor puntualiza de dónde va tomar la acción su algoritmo, enfatizando que es el más importante ya que a partir del diseño de representación, se toman el resto de las acciones que el algoritmo seguirá. Más adelante, señala sus criterios de mutación, las operaciones de cruce y finalmente su criterio de parada. Para la implementación del algoritmo, el autor indica los parámetros que utiliza para el control de su ejecución; también resalta el uso de una lista que representa el estado actual del juego y otra lista que representará el nuevo estado.

El autor finaliza con una conclusión satisfactoria debido a que el prototipo creado ha logrado funcionar y ejecutar, siendo una versión alfa del videojuego. También, da pautas para poder comercializar en un futuro el proyecto (Pérez, 2018).

3.4 Conclusiones

En base a lo revisado en esta sección se concluye lo siguiente:

Aunque se encontró documentación en las distintas bases de datos, no se logró resolver en su totalidad las preguntas planteadas al comienzo del capítulo; esto indica que no existen aún otras soluciones para el problema propuesto, bajo los criterios anteriormente descritos. No obstante, los documentos que fueron seleccionados, brindaron soluciones a considerar para el desarrollo del proyecto.

Una de las soluciones revisadas da mayor enfoque al estándar que se debería de seguir para crear este sistema en los juegos *Gacha*. Cuando se esté en la etapa de obtención de resultados, el algoritmo deberá de tener en consideración cumplir con los aspectos planteados del diseño. Además, se han obtenido soluciones para analizar a los usuarios de estas aplicaciones y determinar si pueden volverse jugadores potenciales, como también prevenir que jugadores tienen riesgo de abandonar el juego.

Por otra parte, al ampliar la investigación, se pudo encontrar que existe la práctica de utilizar algoritmos metaheurísticos para la creación de videojuegos. Sea el caso de programar el comportamiento de la máquina ante las reacciones del jugador, o en el de desarrollo de un videojuego, utilizar el algoritmo propuesto sería útil dentro del contexto establecido.

Sin embargo, no se ha dado una alternativa de solución que se enfoque en el tipo de algoritmo a utilizar para el problema propuesto. Esto hace que la evaluación del estado del arte sea más simple por la escasez de soluciones recolectadas. Al desarrollar este proyecto de fin de carrera se procura aportar al estado del arte actual brindando una nueva forma de solución con un enfoque informático, además de agregar una alternativa de desarrollo para los juegos *Gacha*. Y también, poder incentivar el estudio de estos tipos de juegos en este ámbito.

Capítulo 4. Función objetivo y estructura de datos

4.1 Introducción

El algoritmo propuesto pretende equilibrar la satisfacción del usuario y los ingresos de la empresa en conjunto; para ello, se debe de tomar en cuenta qué variables están relacionadas con ambas partes, como el monto invertido o el peso del premio obtenido. Por tal motivo, en este capítulo se abordarán los dos primeros resultados esperados (R1 y R2) los cuales son la definición de la función objetivo y estructura de datos.

4.2 Función objetivo

Ya que el objetivo es brindar un equilibrio, la función objetivo a proponer consiste en minimizar el valor para que llegue a ser lo más cercano a cero. En el numerador se encuentran las variables que tienen mayor relación con el usuario y en el denominador, las variables relacionadas a la empresa. Finalmente, el valor se disminuye en uno para evaluar que tan cerca se encuentra del equilibrio esperado. De esta forma, la función objetivo sería la siguiente:

$$Premio = \sum_{i=1}^{N} pesoPremio_i \times incurrencyGastada_i$$

$$Monto = \sum_{i=1}^{M} montoInvertido_i \times pesoInversion_i$$

$$Dias = cantDiasSeguidos \times pesoDias$$

$$Incurrency = incurrencyAcumulada \times pesoIncurrency$$

$$F.O. = Mín \left| 1 - \left(\frac{Premio}{Monto + Días + Incurrency} \right) \right|$$

Donde:

- N: Es la cantidad de veces que el usuario ha *rolleado* en el *gacha* del juego.
- M: Es la cantidad de veces que el usuario invirtió en el *gacha* del juego.
- $pesoPremio_i$: Es el valor que tiene el peso del premio i-ésimo obtenido en el *gacha*. Mientras más valioso es el premio, el valor del peso es mayor.
- $incurrencyGastada_i$: Es la cantidad de *incurrency* que se gastó para obtener el premio i-ésimo del *gacha*.
- $montoInvertido_i$: Es la cantidad de dinero real invertido en el *gacha* del juego por i-ésima vez.
- $pesoInversion_i$: Es el valor que tiene el peso de la i-ésima inversión con dinero real en el juego. Mientras mayor es el monto que se invirtió, el valor del peso es mayor.
- $cantDiasSeguidos$: Es la cantidad de días en el que el usuario ha ingresado al juego consecutivamente.
- $pesoDias$: Es el valor que tiene el peso por la cantidad de días consecutivos que el usuario ha ingresado al juego consecutivamente. A mayor cantidad de días, el valor del peso es mayor.
- $incurrencyAcumulada$: Es la cantidad de *incurrency* que el usuario a acumulado a lo largo del juego.
- $pesoIncurrency$: Es el valor que tiene el peso por la cantidad de *incurrency* que el usuario ha acumulado a lo largo del juego. A mayor cantidad de *incurrency*, el valor del peso es mayor.

El uso de la cantidad de días en el denominador de la función se debe a que las empresas toman en cuenta la cantidad en la que los usuarios ingresan a sus juegos para fines estadísticos o también, para brindarles beneficios de acuerdo al día en que ingresaron o por la cantidad

acumulada. En el Anexo B 1, se puede apreciar un ejemplo del funcionamiento de la función objetivo.

4.2.1 Restricciones

Para que la función objetivo sea efectiva, se han considerado las siguientes restricciones:

- **La cantidad de días seguidos debe ser mayor o igual a 1:** Para la empresa, el conteo de días inicia a una hora determinada. Ello implica que, si no logró ingresar al juego antes de dicha hora, se reinicia su contador a 1.

$$cantDiasSeguidos \geq 1$$

- **El monto invertido puede ser igual a 0:** Ya que este tipo de juegos son *free to play*, no siempre los usuarios invertirán dinero real para obtener algún premio.

4.3 Estructura de datos

Para poder implementar el algoritmo *cuckoo search* y el simulador será necesario una estructura de datos que permitirán su ejecución acorde a lo planteado.

4.3.1 Clase Premio

La clase premio almacenará el valor que éste tenga dependiendo de qué tan importante sea. Además, contendrá datos que se actualizarán la información cuando sea obtenido por un usuario que consiste en qué roll se obtuvo y cuánto se gastó para obtenerlo.

Tabla 8: Clase Premio
Fuente: Elaboración propia.

```
Clase Premio {
        int idPremio
        String nombre
        int tipoPremio
        int pesoPremio
        int rollObtenido
        int incurrencyGastada
}
```

Para el caso del simulador, la clase premio se le añadirá el campo *fitness* de tipo *double* para luego compararlo con los del algoritmo.

4.3.2 Clase Usuario

La clase usuario almacenará cuántos días seguidos ha ingresado al juego, el total de *incurrency* tiene el usuario, una lista de monto invertido con dinero real, la cantidad de *rolleos* que ha realizado, una lista de los premios que ha obtenido y los valores que toma el numerador y denominador de la función objetivo.

Tabla 9: Clase Usuario
Fuente: Elaboración propia.

```
Clase Usuario {
        int idUsuario
        int diasSeguidos
        List <Monto> montoInvertido
        int cantRolls
        int vecesInvertidas
        int incurrencyAcumulado
        int idObjetivoPremio
        List <Premio> listaPremios
        double valorUsuario
        double vlaorEmpresa.
}
```

4.3.3 Clase Monto

La clase monto contiene la cantidad de monto invertido y el peso por dicho monto.

Tabla 10: Clase Monto
Fuente: Elaboración propia.

```
Clase Monto {
        double totalMonto
        int peso
}
```

4.3.4 Clase Nido

La clase nido presenta el atributo de la solución, en este caso será el premio con sus datos respectivos. También, se guarda el fitness ya que es un dato que se usa a lo largo de la función.

Esta clase puede ser extendida como una lista de huevos, pero para este proyecto se asumirá el enfoque de los autores del algoritmo donde consideran que cada nido tiene un solo huevo (Yang & Deb, 2009). Con ello, cada nido será una solución óptima.

Tabla 11: Clase Nido
Fuente: Elaboración propia.

```
Clase Nido {
        Premio premio
        double fitness
}
```

4.3.5 Listas

Estas estructuras servirán para representar un conjunto de individuos como una lista de objetos de las clases Premio, Monto y Nido. En el caso de la lista de Nidos, la mejor solución de cada generación, se encontrará en esa lista.

Tabla 12: Ejemplo Listas
Fuente: Elaboración propia.

```
List<Premio> listaPremios
List<Monto> montoInvertido
List<Nido> listaNidos
```

4.3.6 Variables

Además de las clases, se incluirán variables que serán de utilidad para la implementación.

Tabla 13: Ejemplo Variables
Fuente: Elaboración propia.

Variables	Descripción
PESODIAS	Valor del peso por cantidad de días seguidos.
MIN_PESODIAS	Mínimo valor del peso por cantidad de días seguidos.
MAX_PESODIAS	Máximo valor del peso por cantidad de días seguidos.
PESOINCURRENCY	Valor del peso por incurrency acumulada.
MIN_ PESOINCURRENCY	Mínimo valor del peso por incurrency acumulada.
MAX_ PESOINCURRENCY	Máximo valor del peso por incurrency acumulada.
MAX_ITERACIONES	Máximo valor de iteraciones a realizar
MAX_NIDOS	Máximo valor de nidos a evaluar

pa	Ratio de descubrimiento de huevos/soluciones
MAX_TIPOSPREMIO	Máximo valor de tipos de premio en la lista de premios
NUM_ROLL	Valor de rolls hasta el momento
incurrencyAcumulada	Valor de incurrency gastada acumulada hasta el momento

4.4 Conclusión

Como conclusión de esta sección, la complejidad en definir una función objetivo que logre cumplir con el objetivo general del proyecto final de carrera no es alta debido a la identificación y análisis de factores que representan a ambas partes (usuario y empresa). Estos factores se presentan en varios juegos de tipo *gacha* y sirven como referencia para este apartado.

Capítulo 5. Diseño del algoritmo Cuckoo Search

5.1 Introducción

Este capítulo abordará los siguientes resultados esperados (R3 y R4) consisten en el diseño del algoritmo *cuckoo search* en pseudocódigo e implementarlo en el lenguaje de programación propuesto (C#). El algoritmo se basa en las siguientes reglas principales:

1. Cada cuco pone un único huevo cada vez, y lo deja en un nido elegido al azar.
2. Los mejores nidos con mayor calidad del huevo pasarán a la siguiente generación.
3. El número de nidos disponibles en cada generación es fijo, y los huevos descubiertos serán descartados con una probabilidad *pa* del total de nidos, siendo sustituidos por nuevos nidos cada generación.

5.2 Pseudocódigo

En esta sección se presenta el pseudocódigo del algoritmo:

Tabla 14: Pseudocódigo Algoritmo Cuckoo Search
Fuente: Elaboración propia.

```
Inicio CuckooSearch(numGeneraciones,usuario,pa)
    1. listaNidos = CrearNidosIniciales(numNidos);
    2. Para i = 0 hasta numMaxGeneraciones hacer
            a. nido1 = SeleccionarNidoAleatorio(listaNidos)
            b. nido1 = ActualizarViaVueloLevy(nido1)
            c. nido2 = SeleccionarNidoAleatorio(listaNidos)
            d. Si (nido1.fitness < nido2.fitness) entonces
                    i. nido2 = nido1
               Fin Si
            e. RemoverPeoresYGenerarNidos(listaNidos, usuario, pa)
            f. OrdenarMenorAMayorFitness(listaNidos)
       Fin Para
    3. Retornar listaNidos[0]
Fin CuckooSearch
```

Se detalla su secuencia:

- **Línea 1:** Se calcula una lista de nidos iniciales de manera aleatoria.
- **Línea 2:** Se crea una variable **i** para iterar cada generación.

- **Línea 2.a:** Se selecciona un nido aleatorio de la lisa de nidos de la **Línea 1**.
- **Línea 2.b:** Al nido escogido, se le hace una modificación a través de la función de vía vuelos de Levy.
- **Línea 2.c:** Se repite la **Línea 2.a** para escoger otro nido de la lista de nidos.
- **Línea 2.d:** Si el nido modificado es mejor que el otro nido, se reemplaza.
- **Línea 2.e:** Se remueve una fracción pa de peores nidos porque han sido descubiertos.
- **Línea 2.f:** Se ordena la lista de nidos de menor a mayor *fitness*.

❖ **Línea 3:** Se devuelve el mejor nido que está en la primera posición.

5.2.1 Creación de nidos

Para la construcción de nidos iniciales se realiza de forma aleatoria hasta llegar al tamaño de lista de nidos iniciales originales. La consideración inicial es escoger dentro de una lista inicial de premios, uno de los tipos de premios de forma aleatoria y seguidamente, escoger dentro de esa lista, un premio de ese tipo escogido.

Tabla 15: Pseudocódigo Crear Nido Aleatorio
Fuente: Elaboración propia.

```
Inicio CrearNidoAleatorio()
    1. nuevoNido = vacio
    2. tipo = EscogerTipoAleatorio()
    3. nido = AgregarPremioAleatorio(premios,tipo)
    4. Retornar nuevoNido
Fin CrearNidoAleatorio
```

Se detalla su secuencia:

❖ **Línea 1:** Se crea un objeto vacío de tipo Nido.

❖ **Línea 2:** Se escoge un tipo aleatorio de la lista de premios original.

❖ **Línea 3:** Se escoge el premio aleatorio según el tipo escogido y se agrega al nido

❖ **Línea 4:** Se devuelve el nido creado aleatoriamente.

5.2.2 Actualizar vía vuelos de Levy

Se realiza un cambio en la solución seleccionada a base de una distribución de Levy. Para poder hacer dicha simulación, se tomó como referencia el algoritmo de Mantegna, el cual es utilizado por el autor del algoritmo (Yang, 2010).

Tabla 16: Pseudocódigo Actualizar Vía Levy Flights
Fuente: Elaboración propia.

```
Inicio Nido ActualizarViaLevyFlights(nido, usuario, listaPremios)
    1. peso = nido.pesoPremio
    2. beta = 3 / 2;
    3. numSigma = Gamma(1.0 + beta) * Seno(PI * beta / 2.0)
    4. denSigma = Gamma((1.0 + beta) / 2.0) * beta * Potencia(2.0, (beta - 1) / 2.0)
    5. sigma = Potencia (numSigma / denSigma, 1 / beta)
    6. cotaInf = 0
    7. cotaSup = 10
    8. Para i = 1 hasta peso hacer
        a. u = Aleatorio(0, peso) * sigma
        b. v = Aleatorio(0, peso)
        c. Si v == 0 entonces
            i. step = 0
        d. Sino
            i. step = u / Potencia(ValorAbsoluto (v), 1 / beta)
           Fin Si
        e. stepsize = 0.1 * step * (cotaSup - i)
        f. pos = i + stepsize * Aleatorio(0, peso);
        g. pos = DefinirLimites(posStr2, cotaInf, cotaSup)
        h. IntercambiarElementos(nido, listaPremios, pos)
       Fin Para
    9. CalcularFitness(usuario, nido)
    10. Retornar nido
Fin ActualizarViaLevyFlights
```

Se detalla su secuencia:

- **Línea 1 hasta 7:** Se calcula los exponentes y coeficientes para la función, también se declaran las cotas inferior y superior.
- **Línea 8:** Se hace una iteración para cambiar el nido.
  - **Línea 8.a hasta Línea 8.e:** Se hace el cálculo del tamaño del paso.
  - **Línea 8.f:** La posición pos será igual al paso que se dará sobre la lista.
  - **Línea 8.g:** Se acota la variable pos si es demasiado grande o si sale del límite.

- **Línea 8.h:** Se hace el intercambio con el premio de peso pos.

❖ **Línea 9:** Se calcula el *fitness* del nuevo nido.

❖ **Línea 10:** Se retorna el nuevo nido.

5.2.3 Remover y Generar nuevos nidos

Este procedimiento consiste en quitar una parte de los nidos en base a una probabilidad pa, logrando simular que el ave dueña del nido "descubrió" los huevos. En consecuencia, el ave deja el nido y crea uno nuevo.

Tabla 17: Pseudocódigo Remover Nidos y Generar Nuevos
Fuente: Elaboración propia.

```
Inicio RemoverPeoresYGenerarNidos(listaNidos, usuario, pa)
    1. cantNidos = Tamaño(listaNidos)
    2. OrdenarMenorAMayorFitness(listaNidos)
    3. inicio = cantNidos * (1 - p_a))
    4. Para i=inicio hasta cantNidos hacer
        a. listaNidos[i] = CrearNidoAleatorio()
        b. CalcularFitness(usuario, listaNidos[i])
       Fin Para
Fin RemoverPeoresYGenerarNidos
```

Se detalla su secuencia:

❖ **Línea 1:** Se calcula la cantidad de nidos.

❖ **Línea 2:** Se ordena la lista de menor a mayor *fitness*.

❖ **Línea 3:** Se calcula el inicio de donde se removerán una fracción de la lista de nidos.

❖ **Línea 4:** Se itera a partir del valor calculado en **Línea 3**.

- **Línea 4.a:** Se crea un nuevo nido en la posición **i** de la lista de nidos.

- **Línea 4.b:** Se calcula el *fitness* del nuevo nido.

5.2.4 Función de aptitud

Se utilizará la función objetivo definida en la sección 4.2:

$$F.O. = Mín \left| 1 - \left(\frac{Premio}{Monto + Días + Incurrency} \right) \right|$$

La cual es implementada de la siguiente manera:

Tabla 18: Pseudocódigo Calcular Fitness
Fuente: Elaboración propia.

Inicio CalcularFitness(usuario,nido)
1. numerador = 0
2. denominador = 0
3. resultado = 0
4. **Para cada** roll **hasta** cantRollsTotal **hacer**
    a. numerador = numerador + (pesoPremio(roll) *incurrencyGastada(roll))

   Fin Para
5. numerador = numerador + pesoPremioNido*incurrencyAcumulada
6. **Para cada** inversion **hasta** vecesInvertidas **hacer**
    a. denominador = denominador + (montoInvertido(inversion) * pesoMontoInvertido(inversion)

   Fin Para
7. denominador = denominador + (diasSeguidos * pesoDias) + (totalAcumulado * pesoIncurrency)
8. resultado = numerador/denominador
9. **Retornar** ValorAbsoluto(1 - resultado)

Fin CalcularFitness

Se detalla su secuencia:

- **Línea 1 a 3:** Se inicializan las variables acumuladoras que representan los valores del usuario (numerador) y la empresa (denominador).
- **Línea 4:** Se itera para cada **roll** que el usuario ha realizado.
  - **Línea 4.a:** Se acumula el valor resultante del peso del premio del nido multiplicado por la cantidad de *incurrency* que invirtió en dicho **roll** en la variable **numerador**.
- **Línea 5:** Se acumula el valor resultante de **Línea 4.a** con el valor del nuevo premio obtenido por el algoritmo en la variable **numerador**.
- **Línea 6:** Se itera para cada **inversión** que el usuario ha realizado.
  - **Línea 6.a:** Se acumula el valor resultante del monto invertido multiplicado por su peso correspondiente que invirtió en dicha **inversión** en la variable **denominador**.

- **Línea 7:** Se acumula el valor resultante de **Línea 6.a** con el valor de la multiplicación de los días seguidos que el usuario ha entrado al juego por su peso más la multiplicación de *incurrency* acumulada hasta le momento por su peso en la variable **denominador**.
- **Línea 8:** Se almacena en la variable **resultado** el valor de dividir las variables **numerador** y **denominador**.
- **Línea 9:** Se retorna el valor absoluto del cálculo de restar 1 con la variable **resultado**.

5.3 Módulo del algoritmo

Esta sección consiste en explicar los archivos de entrada y salida luego de haber implementado el algoritmo *cuckoo search* en el IDE de Visual Studio en C#.

5.3.1 Archivos de entrada

5.3.1.1 Archivo Premios

Este archivo de texto tendrá las siguientes variables:

- **IdPremio:** El valor identificador del premio.
- **Nombre:** El nombre del premio.
- **TipoPremio:** El valor del tipo del premio.

idPremio	nombre	tipoPremio	pesoPremio
0	Artoria Pendragon	0	9
1	Artoria Pendragon	0	8
2	Nero Claudius	0	8
3	Siegfried	0	7
4	Gaius Julius Caesar	0	1
5	Attila	0	9
6	Gilles de Rais	0	1
7	Chevalier d'Eon	0	7
8	EMIYA	0	7

Figura 7: Archivo Premios
Fuente: Elaboración propia.

- **PesoPremio:** El valor del peso del premio.

5.3.1.2 Archivo Usuarios

Este archivo de texto tendrá las siguientes variables:

- **IdUsuario:** El valor identificador del usuario.
- **DiasSeguidos:** El valor de los días seguidos del usuario al ingresar al juego.
- **CantRolls:** El valor de *rolls* acumulados del usuario.
- **IncurrencyAcumulada:** El valor de *inurrrency* acumulado del usuario.
- **VecesInvertidas:** El valor de inversiones acumuladas del usuario.
- **TotalMonto:** Valor del monto invertido por el usuario **i**.
- **Peso:** Valor del peso del monto invertido **i**.

idUsuario	diasSeguidos	cantRolls	incurrencyAcumulado	vecesInvertidas
1	40	303	11133	4
2	360	70	1200	0
3	1000	700	20333	15
4	30	999	66900	33
5	10	200	5033	4
6	365	90	1217	0
7	200	300	9967	14
8	100	500	9533	10
9	700	800	14133	17
10	800	209	2667	0
totalMonto	peso			
100	10			
300	10			
500	10			
200	10			
totalMonto	peso			
0	0			
totalMonto	peso			
100	10			
50	5			
60	5			
150	10			
100	10			

Figura 8: Archivo Usuarios
Fuente: Elaboración propia.

5.3.1.3 Archivo Premios por Usuario

Este archivo de texto tendrá las siguientes variables:

- **Nombre:** El nombre del premio **j** del usuario **i**.
- **TipoPremio:** El valor del tipo del premio **j** del usuario **i**.
- **PesoPremio:** El valor del peso del premio **j** del usuario **i**.

- **RollObtenido:** El valor del roll en el que se obtuvo el premio **j** del usuario **i**.
- **IncurrencyGastada:** El valor de *incurrency* gastado por el premio **j** del usuario **i**.

nombre	tipoPremio	pesoPremio	rollObtenido	incurrencyGastada
Kaleidoscope	1	5	1	10
Heracles	0	7	2	20
Moony Jewel	1	4	3	30
Bath of the Lunar Goddess	1	6	4	40
Primeval Magic	1	3	5	50
Dragonkind	1	1	6	60
Bath of the Lunar Goddess	1	6	7	70
Imaginary Around	1	5	8	80
Gilles de Rais	0	2	9	90
Moony Jewel	1	4	10	100
Moony Jewel	1	4	1	10
Gilles de Rais	0	2	2	20

Figura 9: Archivo Premios por Usuario
Fuente: Elaboración propia.

5.3.2 Archivos de salida

5.3.2.1 Archivo Resultado CS

Este archivo de texto tendrá las siguientes variables:

CS			
Usuario	830830	Empresa	472160
Roll	Nombre	Peso	Fitness
1	Be Elegant	3	0.797706583
2	Gandr	3	0.794630374
3	Romulus	1	0.791586706
4	Ley Line	1	0.78853187
5	Lancelot	7	0.785639777

Figura 10: Archivo Resultado CS
Fuente: Elaboración propia.

- **ValorUsuario:** Es el valor acumulado final del numerador de la función objetivo.
- **ValorEmpresa:** Es el valor acumulado final del denominador de la función objetivo.
- **Roll:** Es el valor del tiro hecho (Puede ser consecutivo o no).
- **Nombre:** Es el nombre del premio obtenido.
- **Peso:** Es el peso del premio obtenido.
- **Fitness:** Es el *fitness* del premio obtenido.

Se puede apreciar con mayor detalle ejemplos de resultado en el Anexo B 2.

5.4 Conclusión

Como conclusión de esta sección, el poder adaptar el algoritmo metaheurístico al contexto propuesto no implicó mayores dificultades, en base a la teoría y documentación con respecto al algoritmo se pudo lograr el resultado de poder implementar y probar el algoritmo, demostrando que el poder usar este tipo de algoritmo es posible y pueden obtenerse resultados variados.

Capítulo 6. Diseño del simulador

6.1 Introducción

Este capítulo abordará el quinto resultado esperado (R5) el cual consiste en implementar el simulador en el lenguaje de programación propuesto (C#) para luego en otra sección del documento, compararlo con el algoritmo *cuckoo search*. El simulador al cual se hará referencia es de la página GamePress. Este sitio web contiene información de distintos juegos de tipo *gacha* como su descripción, guías, noticias y un simulador. El simulador que se usará para este proyecto es el del juego *Fate/Grand Order* (GamePress, s.f.). Además, se hará énfasis en el proceso de *rolleos* singulares, quiere decir, tiros para obtener premios de uno a la vez.

6.2 Estructura y diseño

6.2.1 Estructura del simulador

Para el simulador se usarán las estructuras planteadas en la sección 4.3 exceptuando la clase Nido. Adicionalmente, se han identificado nuevas listas y variables que permiten su ejecución.

6.2.1.1 Listas premios

A diferencia de la lista de premios del algoritmo *cuckoo search*, el simulador utiliza más sub listas para clasificar los premios con sus respectivos pesos y tipos.

Tabla 19: Ejemplo Listas Simulador
Fuente: Elaboración propia.

```
List<Premio> currFiveStars = new List<Premio>();
List<Premio> currFourStars = new List<Premio>();
List<Premio> currThreeStars = new List<Premio>();
List<Premio> fiveStarBase = new List<Premio>();
List<Premio> fourStarBase = new List<Premio>();
List<Premio> threeStarBase = new List<Premio>();
List<Premio> fiveStarEss = new List<Premio>();
List<Premio> fourStarEss = new List<Premio>();
List<Premio> threeStarEss = new List<Premio>();
List<Premio> currFiveStarEss = new List<Premio>();
List<Premio> currFourStarEss = new List<Premio>();
List<Premio> currThreeStarEss = new List<Premio>();
```

```
List<Premio> currFeatured3S = new List<Premio>();
List<Premio> currFeatured4S = new List<Premio>();
List<Premio> currFeatured5S = new List<Premio>();
List<Premio> currFeatured3E = new List<Premio>();
List<Premio> currFeatured4E = new List<Premio>();
List<Premio> currFeatured5E = new List<Premio>();
```

6.2.1.2 Variables

Además de las variables en la sección 4.3.6, el simulador utiliza variables adicionales para su ejecución. Para los ratios de aparición de peso bajo, son fijos, los demás son basados en cuántos ratios más hay para ese peso.

Tabla 20: Ejemplo Variables Simulador
Fuente: Elaboración propia.

Variables	Descripción
featured5sChance	Valor del ratio de aparición para premios exclusivos de tipo personaje súper raros.
featured4sChance	Valor del ratio de aparición para premios exclusivos de tipo personaje raros.
featured3sChance	Valor del ratio de aparición para premios exclusivos de tipo personaje comunes (Valor fijo).
featured5eChance	Valor del ratio de aparición para premios exclusivos de tipo equipo súper raros.
featured4eChance	Valor del ratio de aparición para premios exclusivos de tipo equipo raros.
featured3eChance	Valor del ratio de aparición para premios exclusivos de tipo equipo comunes (Valor fijo).
rates4s	Valor del ratio de aparición para premios de tipo personaje raros.
rates5s	Valor del ratio de aparición para premios de tipo personaje súper raros.
rates4e	Valor del ratio de aparición para premios de tipo equipo raros.
rates5e	Valor del ratio de aparición para premios de tipo equipo súper raros.

6.3 Pasos del simulador

Para esta sección, se relatará sencillamente el funcionamiento del simulador:

Tabla 21: Secuencia del Simulador
Fuente: Elaboración propia.

Inicio Simulador(usuario)
1. llenarListas()
2. simChanged()

```
3. rarityNum = Aleatorio(N)
4. Si (rarityNum < a) entonces
    a. resultado = pullServant(3)
5. Sino
    a. Si (rarityNum < b) entonces
        i. resultado = pullEssence(4)
    b. Sino
        i. Si (rarityNum < c) entonces
            1. resultado = pullServant (4)
        ii. Sino
            1. Si (rarityNum < d) entonces
                a. resultado = pullServant (5)
            2. Sino
                a. Si (rarityNum < e) entonces
                    i. resultado = pullEssence (3)
                b. Sino
                    i. resultado = pullEssence (5)
                Fin Si
            Fin Si
        Fin Si
    Fin Si
Fin Si
6. resultado.fitness = CalcularFitness(usuario, resultado)
7. Retorno resultado
Fin Simulador
```

Se detalla su secuencia:

- **Línea 1:** Se llenan las listas con sus respectivos premios.
- **Línea 2:** Se separan las listas iniciales con sus respectivos tipos, peso y se obtiene los valores de los ratios.
- **Línea 3:** Se almacena un valor aleatorio en la variable **rarityNum**.
- **Línea 4 hasta Línea 5.b.ii.2.a:** Se compara el valor obtenido de **Línea 3** con un valor fijo, dependiendo de dicho valor, se procede a obtener el premio respectivo.
- **Línea 6:** Se calcula el *fitness* del premio.
- **Línea 7:** Se retorna el premio.

6.4 Módulo del simulador

Esta sección sigue la misma secuencia que la sección 5.3 en donde se explica los archivos de entrada y salida luego de haber implementado el simulador en el IDE de Visual Studio en C#.

6.4.1 Archivos de entrada y salida

Como se explicó en la sección 6.4, se sigue la misma estructura que el algoritmo *cuckoo search*, teniendo los mismos archivos de entrada y salida (Anexo B 2).

Sim			
Usuario	825630	Empresa	472160
Roll	Nombre	Peso	Fitness
1	Euryale	1	0.797662815
2	Cu Chulainn	1	0.794542992
3	The Blue Black Keys	1	0.791455862
4	The Green Black Keys	1	0.788401254
5	Azoth Blade	1	0.785378999

Figura 11: Archivo Resultado Simulador
Fuente: Elaboración propia.

6.5 Conclusión

Como conclusión de esta sección, existió dificultad en poder hallar un simulador con el cual comparar con el algoritmo *cuckoo search*. Debido a que las empresas están en constante competencia comercial, el algoritmo para los *rolleos* no esta disponible para el público o investigadores, así que, para poder hacer una simulación efectiva, se toma en cuenta el porcentaje de aparición de dichos premios e implementarlo luego. Con ello, se puede realizar una simulación efectiva y poder así compararlo.

Capítulo 7. Prototipo de Videojuego

7.1 Introducción

Este capítulo abordará el sexto resultado esperado (R6) el cual consiste desarrollar un prototipo de videojuego de tipo gacha donde se podrá simular la acción de *rollear* por un premio utilizando el algoritmo *cuckoo search* y el simulador.

7.2 Game Design Document (GDD)

Este documento describe las características que contiene un videojuego, desde la descripción de su historia, personajes, mecánicas del juego, audio entre otros elementos. Debido a que este es un prototipo de videojuego, dicho documento no presentará ciertas secciones que no se relacionen con el fin del prototipo. En el Anexo B 3 se encuentra con detalle dicho documento.

7.3 Pantallas

Esta sección presentará las capturas de pantalla del prototipo relacionados con la simulación de *rolleo* en el *gacha*. Las pantallas restantes del prototipo se encuentran en el Anexo B 4.

Figura 12: Pantalla banner de gacha

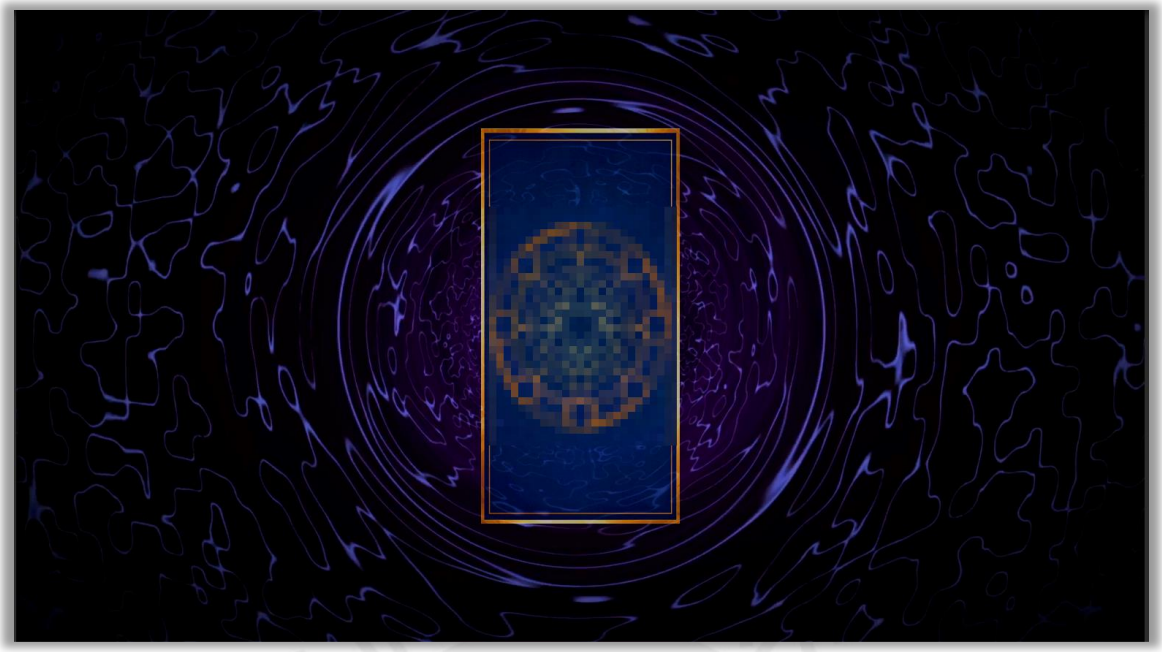


Figura 13: Pantalla premio obtenido (Atrás)

Figura 14: Pantalla premio obtenido

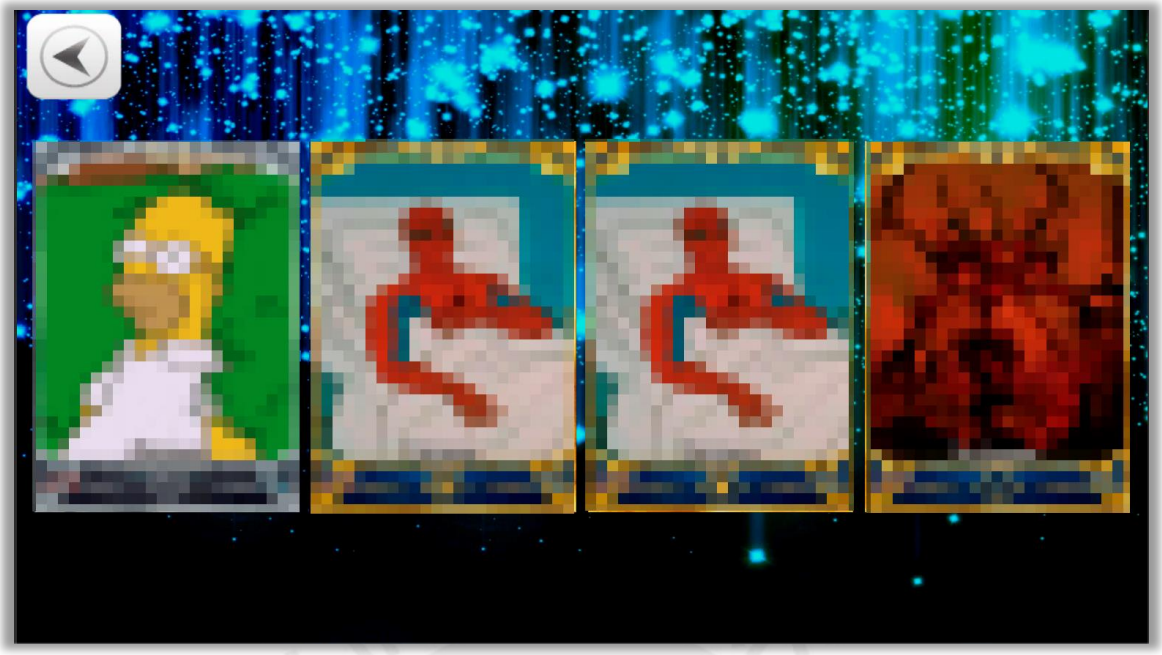


Figura 15: Pantalla historial rolleos

7.4 Conclusiones

Como conclusión de este capítulo, el poder lograr adaptar el algoritmo *cuckoo search* en el prototipo y poder probar su funcionamiento, dio como resultado un gran avance en implementación de juegos de este tipo. También, demuestra que se puede utilizar otros tipos de algoritmo metaheurísticos para poder hacer estas simulaciones con el debido estudio y gusto en desarrollarlo.

Capítulo 8. Experimentación Numérica

8.1 Introducción

Este capítulo abordará el séptimo resultado esperado (R7) el cual consiste en ejecutar el algoritmo *cuckoo search* y el simulador para evaluar su desempeño, teniendo como métrica de comparación el valor de la función objetivo de los resultados.

Ambos casos se ejecutaron 30 veces con 10 muestras distintas. Cada muestra representa a un usuario que ha *rolleado* en el *gacha*. De esas muestras se seleccionan 2; una donde el usuario haya invertido dinero real alguna vez y la otra donde el usuario no haya invertido. Debido a que los resultados de la función objetivo están en un rango de 0 a 1 por la finalidad de minimizar dicho porcentaje, la desviación estándar entre los resultados no afectará en las pruebas estadísticas a realizarse.

8.2 Calibración de Variables

Para realizar efectivamente el objetivo de este capítulo, se debe de justificar los parámetros del algoritmo *cuckoo search* utilizados en este proyecto. En la sección 8.3 se mostrará el resultado final, la ejecución de las pruebas unitarias se encuentran en el Anexo B 5.

8.2.1 Tamaño de nidos

Para hacer efectiva la diferencia de estadísticos, se debe de considerar una cantidad de nidos grande para poder obtener resultados óptimos. Por ello, se considera un tamaño de 100. Esta cantidad se consideró en base a los desarrolladores del algoritmo original (Yang & Deb, 2009) y por pruebas unitarias (Ver Anexo B 5).

8.2.2 Probabilidad pa

Esta variable es importante para que en la ejecución del algoritmo *cuckoo search* varíe por iteración. Esta variable tiene contiene una porción de los peores nidos que se van a

descartar. Para este proyecto, se escogió un valor de 0.4 debido al cambio de resultados obtenidos en pruebas unitarias (Ver Anexo B 5) y también tomando referencia en otras implementaciones (Valian, Mohana & Tavakoli, 2011).

8.2.3 Cantidad de generaciones

El número total de generaciones se interpretan en el algoritmo como el número de iteraciones que se ejecutará. Se consideró este parámetro debido a los resultados obtenidos dentro de las pruebas unitarias con valores de 500, 1000 (valor original) y 2000.

Nidos	Iteraciones	pa	
100	1000	0.4	
Resultado			
Usuario1	828810	Empresa	472160
1	Gaius Julius Caesar	1	0.79766282
2	Mephistopheles	1	0.79454299
3	Gilles de Rais	2	0.79147767
4	Imaginary Number Magecraft	3	0.7885101
5	Elizabeth Bathory	7	0.78574844
6	Angel's Song	3	0.78332176
7	Mephistopheles	1	0.78057866
8	Projection	3	0.7778018
9	Imaginary Number Magecraft	3	0.77525035
10	Mephistopheles	1	0.7730524
11	Attila	9	0.77051789
12	Gilles de Rais	2	0.76945208
13	The Red Black Keys	1	0.76729681
14	Primeval Magic	3	0.76459261
15	Bath of the Lunar Goddess	6	0.76253845
16	The Blue Black Keys	1	0.76134425
17	The Green Black Keys	1	0.75868529
18	Gandr	3	0.75609912
19	Sakata Kintoki	10	0.75441276
20	Ushiwakamaru	1	0.75527364

Figura 16: Prueba Calibración de Variables
Fuente: Elaboración propia.

Luego de realizar dichas ejecuciones, se observó que para el resultado original (1000) era el más óptimo a comparación de los otros 2. Con ello, estas pruebas justificaron la razón de utilizar dicha cantidad (Ver Anexo B 5).

8.3 Generación de las muestras

Para la creación de las muestras, se utilizó funciones aleatorias para generar los premios que contiene cada usuario, teniendo también en consideración que el tipo de premio obtenido sea acorde con el porcentaje de aparición que el simulador (Ver Sección 6).

También, para los datos de cada usuario, se utilizó estas funciones para completar los datos de inversión y atributos que este necesita para poder ser utilizado en el algoritmo. Además, el tamaño inicial de la lista de premios fue tomada como referencia del juego Fate/Grand Order en su etapa inicial (72 premios entre personajes y equipamiento).

8.4 Resultados

En esta sección del capítulo se muestran los resultados obtenidos luego de ejecutar el algoritmo y el simulador. Se muestran para el caso del usuario que invierte (Tabla P2W) y el usuario que no invierte (Tabla F2P).

Tabla 22: Ejecución muestras usuario que invirtió
Fuente: Elaboración propia.

P2W		
Prueba	**CS**	**Simulador**
1	0.783127273	0.783466667
2	0.783723738	0.784110118
3	0.784267501	0.784700505
4	0.784567457	0.785382219
5	0.784053473	0.786010981
6	0.784542212	0.786539834
7	0.784600805	0.786827766
8	0.784989379	0.787302337
9	0.784387491	0.787585234
10	0.781775591	0.786741626
11	0.781983664	0.786091015
12	0.780888166	0.784933736
13	0.781051656	0.784618948
14	0.77978306	0.78421417
15	0.775166705	0.783536445
16	0.775028637	0.780252005
17	0.770737275	0.777585026
18	0.766431658	0.777621105
19	0.766054838	0.777294358
20	0.758726575	0.771505983

21	0.758695163	0.77048369
22	0.758619144	0.768572069
23	0.758498995	0.76841635
24	0.758068106	0.768172713
25	0.751474828	0.766910623
26	0.744486188	0.766585635
27	0.734155472	0.765161859
28	0.729163924	0.764801404
29	0.724163569	0.764399738
30	0.723774243	0.763957289

Tabla 23: Ejecución muestras usuario que no invirtió
Fuente: Elaboración propia.

F2P		
Prueba	**CS**	**Simulador**
1	0.714655172	0.715086207
2	0.71779661	0.719491525
3	0.719583333	0.723333333
4	0.720491803	0.726639344
5	0.718548387	0.729435484
6	0.708333333	0.731746032
7	0.693359375	0.73359375
8	0.685384615	0.735
9	0.681818182	0.735984848
10	0.683208955	0.736567164
11	0.684191176	0.736764706
12	0.68442029	0.736594203
13	0.6775	0.736071429
14	0.636619718	0.733450704
15	0.599305556	0.709722222
16	0.578082192	0.708219178
17	0.541216216	0.701689189
18	0.512333333	0.7
19	0.467434211	0.696381579
20	0.456493506	0.664935065
21	0.457051282	0.6625
22	0.45664557	0.653797468
23	0.4421875	0.65125
24	0.414814815	0.648148148
25	0.414634146	0.638109756
26	0.414156627	0.633433735
27	0.411607143	0.592857143
28	0.363823529	0.589705882
29	0.337209302	0.586046512
30	0.271264368	0.573563218

8.5 Prueba Kolmogorov-Smirnov

Para poder realizar la Prueba Z se debe de verificar que las muestras sigan una distribución normal. Por ello, se debe de demostrar que el resultado obtenido al ejecutar el algoritmo y el simulador muestran dicha distribución.

8.5.1 Prueba para el algoritmo Cuckoo Search

Se plantean las hipótesis para el caso:

- H0: Los valores de la muestra del algoritmo *cuckoo search* presentan una distribución normal.
- H1: Los valores de la muestra del algoritmo *cuckoo search* no presentan una distribución normal.

Se realiza la prueba:

Tabla 24: Resultado de prueba Kolmogorov-Smirnov para el algoritmo Cuckoo Search
Fuente: Elaboración Propia

TIPO	Media	Desviación Estándar	Valor Crítico (D)	p-value
P2W	0.7672329	0.01960249	0.20566	0.1369
F2P	0.562139	0.1410651	0.22659	0.07784

Con una significancia de 0.05, para ambos casos aceptamos la hipótesis nula (H0), demostrando que los datos presentan una distribución normal.

8.5.2 Prueba para el simulador

Se plantean las hipótesis para el caso:

- H0: Los valores de la muestra del simulador presentan una distribución normal.
- H1: Los valores de la muestra del simulador no presentan una distribución normal.

Se realiza la prueba:

Tabla 25:Resultado de prueba Kolmogorov-Smirnov para el Simulador
Fuente: Elaboración propia.

TIPO	Media	Desviación Estándar	Valor Crítico (D)	p-value
P2W	0.7777927	0.008745661	0.24176	0.04992
F2P	0.6880039	0.0526558	0.19654	0.1721

Con una significancia de 0.05, para ambos casos aceptamos la hipótesis nula (H0), demostrando que los datos presentan una distribución normal.

8.6 Prueba F de Fisher

La siguiente condición para poder realizar la Prueba Z consiste en que el algoritmo y el simulador sean significativamente homogéneas.

Se plantean las hipótesis para el caso:

- H0: Las varianzas del algoritmo *cuckoo search* y el simulador son significativamente homogéneas
- H1: Las varianzas del algoritmo *cuckoo search* y el simulador son significativamente diferentes.

Se realiza la prueba:

Tabla 26: Resultado de prueba F de Fisher
Fuente: Elaboración propia.

TIPO	F	Valor Mínimo	Valor Crítico	p-value
P2W	5.0239	2.39118	10.55511	4.025e-05
F2P	7.1771	3.416029	15.078977	9.007e-07

Con una significancia de 0.05, ambos casos son menores y con ello se rechaza la hipótesis nula (H0).

8.7 Prueba Mann-Whitney-Wilcoxon

En base a los resultados obtenidos de la prueba F de Fisher, no se puede realizar la prueba Z ya que las muestras presentan varianzas no significativamente homogéneas y con ello, no

poder realizar adecuadamente pruebas paramétricas. Por ello, se realizará la prueba de Mann-Whitney-Wilcoxon para pruebas no paramétricas como alternativa y lograr así comparar las medias.

Se plantean las hipótesis para el caso:

- H0: La media del algoritmo *cuckoo search* es mayor que la media del simulador.
- H1: La media del algoritmo *cuckoo search* es menor que la media del simulador.

Se realiza la prueba:

Tabla 27: Resultado de prueba Mann-Whitney-Wilcoxon
Fuente: Elaboración propia

TIPO	W	Valor Mínimo	Valor Crítico	P-value
P2W	280	-0.013372843	-0.00099472	0.01147
F2P	188	-0.19715190	-0.03222222	6.264e-05

Para una significancia de 0.05, en ambos casos se rechaza la hipótesis nula (H0), demostrando que las medias son distintas y también, demostrar que la media del algoritmo *cuckoo search* es menor que la del simulador.

Como conclusión de esta sección, se aprecia que el algoritmo *cuckoo search* tiene un mejor desempeño que el simulador al poder cumplir con el objetivo de la función objetivo en minimizar su valor y acercarse al valor de 0, asegurando que los *rolleos* obtenidos con el algoritmo sean más probables de obtener mejores recompensas.

Capítulo 9. Conclusiones y trabajos futuros

9.1 Conclusiones

Después de revisar documentos relacionados, realizar los experimentos trazados y obtener el resultado de los mismos, se aprecia que los valores de la función objetivo van disminuyendo a medida que el algoritmo o el simulador se ejecuta continuamente. Esto cumple con lo propuesto, minimizando el valor representativo del usuario (valor del premio obtenido) y el beneficio de la empresa (monto de dinero invertido).

Asimismo, el adaptar el algoritmo *cuckoo search* dentro del contexto de los juegos *gacha* ha sido posible pese al no haber documentación ni investigaciones previas, demostrando tanto su aplicación viable, como la posibilidad de adecuar algoritmos metaheurísticos en un entorno diferente. De igual manera, el poder aplicar la funcionalidad del algoritmo en un prototipo de juego, demuestra que sus ejecuciones logran simular con éxito un videojuego de tipo gacha.

También, evaluando los resultados de la experimentación numérica, se comprueba que el valor de la función objetivo va disminuyendo a medida que se realizan más pruebas para el algoritmo y el simulador. Logrando demostrarse un mayor desempeño del algoritmo frente al simulador. Por otra parte, se podría mejorar aún más el algoritmo si se llegara a conseguir que los datos sean paramétricos y realizar la prueba Z. Un punto de observación puede ser dentro del mismo algoritmo en el procedimiento de remover y generar nuevos nidos, donde puede que se eliminen mejores alternativas y con ello, conseguir mejores resultados.

Finalmente, ha sido efectivo el diseñar el algoritmo *cuckoo search* y el simulador en el presente proyecto de fin de carrera para encontrar el equilibro entre la satisfacción del cliente y el beneficio de la empresa. Entonces, es factible y beneficioso que nuevos desarrolladores o empresas puedan aplicar este algoritmo como base de su juego *gacha*.

9.2 Trabajos futuros

Se proponen los siguientes trabajos futuros a partir de este proyecto de fin de carrera:

- Obtener el algoritmo/función/procedimiento de un videojuego de tipo gacha de una empresa desarrolladora para hacer más efectiva la comparación con el algoritmo.
- Modificar el algoritmo *Cuckoo Search* para mejorar la solución óptima y poder competir y superar el simulador.
- Incluir en el algoritmo un procedimiento para poder hacer *rolleos* en grupos de 10. El objetivo a alcanzar era el de poder obtener un premio mediante *rolleos*, mas no se incluyó el poder hacer *rolleos* grupales, por ello, se espera poder hacer inclusión de esta mecánica.

Referencias

Batchelor, J. (2018). GamesIndustry.biz presents... The Year In Numbers 2018.

Recuperado el 4 de abril del 2019, a partir de

https://www.gamesindustry.biz/articles/2018-12-17-gamesindustry-biz-presents-the-year-in-numbers-2018

Nelson, R. (2016). Sony's Fate/Grand Order Hits $3 Billion in Global Player

Spending.

Recuperado el 22 de abril del 2019, a partir de

https://sensortower.com/blog/fate-grand-order-revenue-3-billion

Apple Support Page.

Recuperado el 8 de abril del 2019, a partir de

https://support.apple.com/en-us/HT202023

Fire Emblem Heroes FAQ Page.

Recuperado el 8 de abril del 2019, a partir de

https://support.fire-emblem-heroes.com/en-US/faq

Honkai Impact 3rd About Page.

Recuperado el 21 de abril del 2019, a partir de

http://www.global.honkaiimpact3.com/system/about.html

miHoYo. (2016). Honkai Impact 3rd (Versión 3.0.0*) [Aplicación Móvil].*

Recuperado el 22 de abril del 2019. Descargado de

https://play.google.com/store/apps/details?id=com.miHoYo.bh3global

Baglin, S. (2017). Random Numbers & Gaming.

Recuperado el 4 de abril del 2019, a partir de

https://scholarworks.sjsu.edu/art108/7/

Aniplex. (2017). Fate/Grand Order (Versión 1.27.1*) [Aplicación Móvil].*

Recuperado el 21 de abril del 2019. Descargado de

https://play.google.com/store/apps/details?id=com.aniplex.fategrandorder.en

Nintendo. [Nintendo Mobile]. (2019, marzo 27). Fire Emblem Heroes - Mythic Hero

(Yune: Chaos Goddess) [Archivo de video].

Recuperado el 6 de abril del 2019, de

https://www.youtube.com/watch?v=4WTtsrJxRBA

Nakamura, Y. (2018, noviembre 5). The good times are over for Japan's Loot-Box-Style

Gaming Bonanza

Recuperado el 27 de agosto del 2019, de

https://www.bloomberg.com/news/articles/2018-11-05/the-good-times-are-over-for-japan-s-loot-box-style-gaming-bonanza

Cowley, B., Charles, D. Black, M., and Hickey, R. (2008) Toward and understanding of

flow in video games. ACM Comput. Entertain. 6, 2, Article 20 (July 2008), 27 Pages.

DOI = 10.1145/1371216.1371223

Recuperado el 20 de setiembre del 2019, a partir de

http://doi.acm.org/10.1145/1371216.1371223

De Vere, K. (2012). Japan officially declares lucractive kompu gacha practice illegal in social games.

Recuperado el 5 de junio del 2019, a partir de https://www.adweek.com/digital/japan-officially-declares-lucractive-kompu-gacha-practice-illegal-in-social-games/

McKenzie, B. (2018). Loot Boxes in Japan: Legal Analysis and Kompu Gacha Explained.

Recuperado el 5 de junio del 2019, a partir de https://www.lexology.com/library/detail.aspx?g=9207df10-a8a2-4f67-81c3-6a148a6100e2

Muhammad, K., Gao, S., Qaisar, S., Abdul, M. Muhammad, A., Usman, A., Aleena, A. & Shadid, A. (2018). Comparative Analysis of Meta-Heuristic Algorithms for SolvingOptimization Problems.

Recuperado el 25 de abril del 2019, a partir de https://doi.org/10.2991/meici-18.2018.121

Mohamad, A., Zain, A. & Nazira, N. (2014). Cuckoo Search Algorithm for Optimization Problems—A Literature Review and its Applications.

Recuperado el 25 de abril del 2019, a partir de https://doi.org/10.1080/08839514.2014.904599

Yang, XS. & Deb, S. (2014). Cuckoo Search via Lévy flights.

Recuperado el 5 de junio del 2019, a partir de

https://doi.org/10.1007/s00521-013-1367-1

PSeInt. (2019). ¿Para qué sirve PSeInt?.

Recuperado el 17 de junio del 2019, a partir de

http://pseint.sourceforge.net/index.php?page=features.php

MICROSOFT. (2019). Le damos la bienvenida al IDE de Visual Studio.

Recuperado el 28 de mayo del 2019, a partir de

https://docs.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2019

MICROSOFT. (2015). Introducción al lenguaje C# y .NET Framework.

Recuperado el 28 de mayo del 2019, a partir de

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/language-specification/introduction

Unity. (2019). Explore Unity.

Recuperado el 28 de mayo del 2019, a partir de

https://unity3d.com/unity

Android Studio. (2017). Meet Adnroid Studio User Guide.

Recuperado el 5 de junio del 2019, a partir de

https://developer.android.com/studio/intro

Box, G., Hunter, W., & Hunter, J. (1993). Estadística para investigadores.

España, Barcelona: Reverté.

García, R., Gonzáles, J. & Jornet, J. (2010). SPSS Pruebas no paramétricas.

España, Valencia: Grupo de Innovación educativa.

Lin, C. & Mudholkar, G. (1980). A simple test for normality against asymetric

alternatives.

Recuperado el 28 de mayo del 2019, a partir de

https://www.jstor.org/stable/2335489

Servisoftcorp. (2019). Definición y cómo funcionan las aplicaciones móviles.

Recuperado el 8 de abril del 2019, a partir de

https://www.servisoftcorp.com/definicion-y-como-funcionan-las-aplicaciones-moviles/#

Real Academia Española [RAE]. (2019). Significado de la palabra "videojuego"

Recuperado el 8 de abril del 2019, a partir de

https://dle.rae.es/?id=bmnbNU7

Gómez, L. (2018). Del hábito cotidiano a la profesionalización entre video – jugadores

en línea: la interpretación de los video-juegos como contenido audiovisual para

jugadores de Lima Metropolitana en la década del 2010 (tesis de pregrado).

Pontificia Universidad Católica del Perú, Lima, Perú.

Tack, D. (2013). The Subscription Transition: MMORPGs And Free-To-Play.

Estados Unidos de América: Forbes.

Recuperado el 8 de abril del 2019, a partir de

https://www.forbes.com/sites/danieltack/2013/10/09/the-subscription-transition-mmorpgs-and-free-to-play/#14ef24ee2c35

Real Academia Española [RAE]. (2019). Significado de la palabra "simulador"

Recuperado el 4 de junio del 2019, a partir de

https://dle.rae.es/?id=Xw14yph

Famularo, J. (2017). 'Fire Emblem Heroes' Is a Gacha Game – Here's What That Means'. Estados Unidos de América: Inverse.

Recuperado el 8 de abril del 2019, a partir de

https://www.inverse.com/article/27267-what-are-gacha-games-fire-emblem-heroes

Curator EN. (2014). Gaming Currency.

Estados Unidos de América: Community Currency Knowledge Gateway.

Recuperado el 8 de abril del 2019, a partir de

http://community-currency.info/en/glossary/gaming-currency/

Brassard, G. & Bratley, P. (1998). Fundamentos de Algoritmia.

España, Madrid: Prentice Hall.

Real Academia Española [RAE]. (2019). Significado de la palabra "heurística".

Recuperado el 27 de abril del 2019, a partir de

https://dle.rae.es/?id=KHdGTfC

González, P., Lorés, J. & Pascual, A. (2006). Evaluación Heurística. En Lorés, J. (Ed),

La Interacción Persona Ordenador (pp 1-40). AIPO Press (electronic book), ISBN 84-607-2255-4.

Vélez, M. & Montoya, J. (2007). Metaheurísticos: Una alternativa para la solución de problemas combinatorios en administración de operaciones. En Revista EIA, (8), 99-115.

Recuperado el 27 de abril del 2019, a partir de http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1794-12372007000200009&lng=en&tlng=es

Yang, XS. & Deb, S. (2009). Cuckoo Search via Lévy flights.

Recuperado el 27 de abril del 2019, a partir de https://doi.org/10.1109/NABIC.2009.5393690

Schmitt, L. (2001). Theory of genetic algorithms.

Theoretical Computer Science.

Kuri, A. & Galaviz, K. (2007). Algoritmos genéticos.

México, México D.F: Sociedad Mexicana de Inteligencia Artificial.

Pocket Gamer. [PocketGamerbiz]. (2017, octubre 24). The recipe for strong gacha.

Recuperado el 21 de abril del 2019, a partir de

https://www.youtube.com/watch?v=_81mVrq1_lk

Sifa, R., Hadiji, F., Runge, J., Drachen, A., Kersting, K. & Bauckhage, C. *(2015).*

Predicting Purchase Decisions in Mobile Free-to-Play Games.

Recuperado el 22 de abril del 2019, a partir de

https://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/viewPaper/11544

Runge, J., Gao, P., Garcin, F. & Faltings, B. *(2014). Churn Prediction for High-Value Players in Casual Social Games.*

Recuperado el 22 de abril del 2019, a partir de

https://ieeexplore.ieee.org/abstract/document/6932875

Recio, G., Martin, E., Estébanez, C. & Saez, Y. (2012). AntBot: Colonies for Video Games

Recuperado el 25 de abril del 2019, a partir de

https://ieeexplore.ieee.org/document/6262464

Pérez, J. (2018). Videojuego RPG (Role-Playing Game) basado en algoritmos evolutivos.

Recuperado el 25 de abril del 2019, a partir de

http://tauja.ujaen.es/jspui/handle/10953.1/8447

Game Express (2018). Summon Simulator

Recuperado el 20 de setiembre de 2019. a partir del

https://grandorder.gamepress.gg/summon-simulator

Valian, E., Mohanna, S., & Tavakoli, S. (2011). Improved cuckoo search algorithm for global optimization. International Journal of Communications and Information Technology, 1(1), 31–44.

Las imágenes utilizadas en el prototipo se obtuvieron de:

Unplash (2019)

Recuperado el 24 de noviembre de 2019. a partir del

https://grandorder.gamepress.gg/summon-simulatorhttps://unsplash.com/

Las pistas de música utilizadas en el prototipo se obtuvieron de:

Pixabay (2019)

Recuperado el 21 de noviembre de 2019. a partir del

https://pixabay.com/music/

Los videos utilizados en el prototipo se obtuvieron de:

Pexel (2019).

Recuperado el 23 de noviembre de 2019. a partir del

https://www.pexels.com/videos

Anexo

Anexo A: Plan de Proyecto

1. Justificación

Los juegos de tipo *Gacha* son una alternativa de negocio para empresas desarrolladoras de juegos, tanto para emprendedoras como para las más reconocidas. Estos juegos pueden llegar a ser muy exitosos, pero también, puede darse el caso de un declive económico al no poder conservar a sus usuarios (De Vere, 2012).

Para los jugadores, invertir una gran suma de dinero real sin obtener lo que desean puede provocar reclamos y demandas a estas compañías, como consecuencia esta situación incentiva a que estos juegos sean más del tipo de azar, especialmente si los consumidores son menores de edad (De Vere, 2012). De ser así, los resultados a producirse van desde el grado de exigencia de las leyes reguladoras, hasta la prohibición del consumo de estos juegos.

Actualmente existen soluciones parciales para el problema planteado de complacer al usuario sin que las empresas pierdan ganancias, pero son solo temporales o no llegan a cumplir las expectativas de los agentes involucrados en su totalidad (McKenzie, 2018). Por esta razón, el presente proyecto de fin de carrera promete brindar una alternativa distinta y moderna que no se haya planteado anteriormente. El uso del algoritmo propuesto favorece a las empresas desarrolladoras en gran medida dentro de la parte económica al lograr mantener a sus usuarios y hacer que inviertan en sus juegos; pero sin descuidar la parte publicitaria al conseguir que sus jugadores den opiniones positivas de este y recomendarlo a otras personas. Además, dicho algoritmo ha demostrado tener un mejor desempeño que otros en distintas aplicaciones ya sea en problemas de programación de horarios, o en resolver problemas de cálculo.

Dicho todo esto, este proyecto de fin de carrera espera complementar y motivar al estudio e investigación de estos juegos y sus derivaciones. También se tiene la expectativa de que futuras generaciones logren alcanzar la respuesta deseada respecto al equilibrio adecuado.

2. Viabilidad

1.1. Viabilidad técnica

En el presente proyecto de fin de carrera se utilizarán herramientas conocidas por el autor, entre estas el lenguaje C# y el IDE de Visual Studio que se han utilizado a lo largo de proyectos y estudios en la carrera de Ingeniería Informática. Además, se cuenta con que estas herramientas son fáciles de obtener tanto en las páginas web oficiales, como también en los equipos de los laboratorios de la universidad. Igualmente se cuenta con la orientación del asesor, quien tiene valiosa experiencia en los temas de algoritmos.

De igual modo, para el diseño del videojuego, las herramientas de Unity y Android Studio tienen versiones gratuitas para su descarga en sus páginas oficiales. Al considerar en el alcance el diseño de un videojuego no tan complejo, no representará mucha dificultad al momento del desarrollo. Sin embargo, el autor tiene un conocimiento básico en el uso de estas herramientas, pero no produce un riesgo en el desarrollo del proyecto al haber documentación en línea sobre el uso de estas.

Con todo lo mencionado, se concluye que en el aspecto técnico no existirán obstáculos mayores que influyan en el desarrollo del proyecto.

1.2. Viabilidad temporal

Para el desarrollo de este proyecto, se dispone de un tiempo limitado de 4 meses, el cual es el tiempo total de un ciclo regular de universidad. Para garantizar la culminación total del proyecto, se seguirá el plan de proyecto (Ver Anexo 8) a partir de la fase de ejecución. Dicho plan de proyecto comenzará luego de culminar el curso de Proyecto de Tesis 1.

1.3. Viabilidad económica

Este proyecto es de carácter académico, por lo que no tiene fines de lucro. Las herramientas a utilizar son gratuitas y se encuentran disponibles en sus páginas respectivas o en los equipos de la universidad. Además, se cuenta con una laptop que posee las características necesarias para el desarrollo satisfactorio de este proyecto y proveer de resultados acordes a lo esperado.

1.4. Conclusiones

En conclusión, se cuenta con los conocimientos necesarios, las herramientas disponibles y la orientación del asesor para comenzar el proyecto de fin de carrera. Así mismo, si hubiera algún percance en las herramientas a utilizar, estas también se encuentran instaladas en los equipos de la universidad. Por último, como se puede observar en el cronograma, es posible culminar en el tiempo determinado. Con todo esto, se puede decir que el proyecto de fin de carrera es completamente viable.

3. Alcance

El presente proyecto tiene como objetivo brindar una solución que abarque la satisfacción del usuario y los ingresos de las empresas desarrolladoras de los juegos con sistemas de premios en conjunto, esto se realizará con la implementación de un algoritmo metaheurístico *cuckoo search*. Para alcanzar dicho objetivo, se tendrá en consideración ciertas variables para encontrar el equilibro entre las dos partes. Las variables a considerar son: la cantidad de días consecutivos en entrar al juego, la cantidad acumulada de *incurrency* invertida al momento de *rollear*, la cantidad acumulada de personajes/objetos conseguidos durante los *rolls*, el costo(peso) del personaje/obtenido y si el usuario ha invertido dinero real en el juego.

El algoritmo *cuckoo search* brinda soluciones a problemas de análisis y estabilidad frente a otros algoritmos, demostrando ser más eficiente. Adicionalmente, se ha utilizado este

algoritmo en distintas áreas de optimización e inteligencia computacional. Para su implementación se definirán la función y la estructura de datos que se empleará, seguidamente se construirá el pseudocódigo para posteriormente llevarlo al lenguaje de programación escogido; este mismo proceso se hará con el simulador. Finalmente, al tener ambos algoritmos, se realizará la experimentación numérica con el propósito de validar el desempeño del algoritmo cuco frente al competidor, comprobando que puede ser una alternativa de solución.

Además, se desarrollará un prototipo de videojuego simple en donde se mostrará visualmente el funcionamiento del algoritmo a implementar. Este prototipo se enfocará en la dinámica de *rollear*, donde al realizar la acción (en el juego), el algoritmo mostrará al usuario el resultado obtenido.

Cabe señalar que los videojuegos usados como referencia para desarrollar lo dicho, serán los juegos *Fate/Grand Order* y *Fire Emblem Heroes*. Esto se debe al previo conocimiento, uso y las múltiples experiencias de *rollear* en estos juegos.

4. Limitaciones

Para el desarrollo del proyecto de fin de carrera, se consideraron las siguientes limitaciones:

- El no poder conocer u obtener la familia del algoritmo que utilizan estos tipos de juegos, se utilizará un simulador a base de pruebas y experiencias para poder compararlo con los resultados que muestre el algoritmo escogido.
- El tiempo de ejecución de los algoritmos dependerá del hardware y software de la computadora en la cual se ejecutará.

5. Riesgos

A continuación, se presentará en la siguiente tabla (ver Tabla 3) los riesgos que podrían impactar en el desarrollo del proyecto.

Tabla A1: Tabla de riesgos

Riesgo	Impacto	Estrategia de mitigación
Pérdida de la información parcial o total del proyecto.	Alta: Presentación de entregables fuera de tiempo. Se pierde tiempo en recuperarla y alta probabilidad de reprobar el curso.	Realizar respaldos del proyecto con frecuencia y tenerlo en diferentes almacenamientos como USB o en la nube (Google Drive, DropBox).
Planeamiento ineficaz del proyecto.	Alto: Fallo de presentar los entregables. Acumulamiento de trabajo.	Revisar los plazos de entrega para reorganizar la presentación de entregables.

6. Estructura de descomposición del trabajo (EDT)

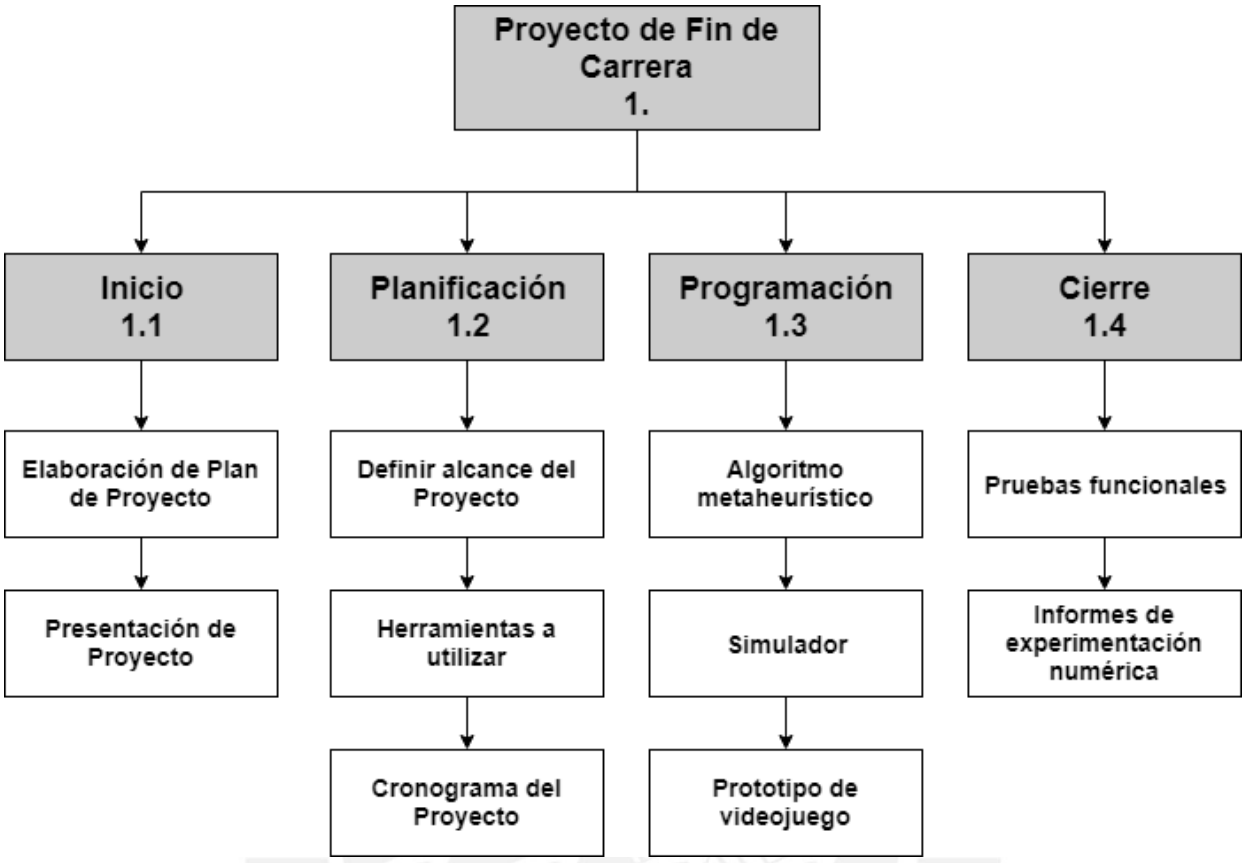


Figura A1: EDT

7. Lista de tareas

	Nombre de la tarea	Descripción	Duración	Inicio	Finalizar
1	PROYECTO DE FIN DE CARRERA		217d	18/03/19	25/11/19
2	Implementación de un algoritmo metaheurístico Cuckoo Search, para sistemas de premiación de juegos		217d	18/03/19	25/11/19
3	1. Fase de Planeación		96d	18/03/19	06/07/19
4	1.1. Problemática		42d	18/03/19	04/05/19
5	Definición del problema	Se realiza la investigación del problema planteado en base a artículos, noticias, documentos, entre otros.	19d	18/03/19	08/04/19
6	Investigación del estado del arte	Se realiza la búsqueda de posibles soluciones existentes en bases de datos de páginas de investigación y tesis.	23d	09/04/19	04/05/19
7	1.2. Marco conceptual y objetivos		26d	06/05/19	04/06/19
8	Desarrollo del marco conceptual	Se hace la búsqueda de las definiciones que no han sido explicadas en la problemática.	5d	06/05/19	10/05/19
9	Definición del objetivo general y específicos	Se evalúa la información recolectada y se definen los objetivos para resolver el problema.	21d	11/05/19	04/06/19
10	1.3. Resultados y herramientas		14d	05/06/19	20/06/19
11	Definición de resultados esperados	En base a los objetivos propuestos, se analizan los posibles resultados para cada uno de los objetivos específicos.	8d	05/06/19	13/06/19
12	Investigación sobre herramientas	Se hace una búsqueda de las herramientas a utilizar para lograr hacer el desarrollo del proyecto.	6d	14/06/19	20/06/19
13	1.4. Alcance, limitaciones, viabilidad		14d	21/06/19	06/07/19
14	Definición de alcance y limitaciones	Se hace un análisis para revisar hasta donde el proyecto va a llegar y las limitaciones.	4d	21/06/19	25/06/19
15	Elaboración de la justificación del proyecto	En base a todo lo investigado y planteado, se hace una justificación de porqué el desarrollo del proyecto.	4d	26/06/19	29/06/19
16	Análisis de riesgos y de viabilidad	Se revisa y busca los posibles riesgos que puedan ocurrir y qué tan viable es el proyecto.	6d	01/07/19	06/07/19
17	2. Fase de Ejecución		86d	19/08/19	26/11/19
18	2.1. Iteración 1		35d	19/08/19	27/09/19
19	Diseñar función que permita la estimación entre usuarios y empresa	Se definen las variables a utilizar en la función y luego se crea la función.	11d	19/08/19	30/08/19
20	Estructura de datos que soporten la implementación del algoritmo Cuckoo Search	Se buscan los datos que se utilizarán en los procedimientos de ejecución del algoritmo.	5d	31/08/19	05/09/19
21	Algoritmo Cuckoo Search diseñado	Teniendo la función, se diseña el pseudocódigo del algoritmo.	7d	06/09/19	13/09/19
22	Módulo para carga de datos del algoritmo Cuckoo Search	Con el algoritmo diseñado, se implementa para su uso con los datos seleccionados.	12d	14/09/19	27/09/19
23	2.2. Iteración 2		22d	28/09/19	23/10/19
24	Adaptar función en el simulador	Con la función previamente diseñada, se adapta al simulador escogido.	7d	28/09/19	05/10/19
25	Estructuras de datos que soporten el simulador	Se buscan los datos que se utilizarán en los procedimientos de ejecución del simulador.	4d	07/10/19	10/10/19
26	Simulador diseñado	Teniendo la función, se diseña el simulador.	3d	11/10/19	14/10/19
27	Módulo para la carga de datos del simulador	Con el simulador diseñado, se implementa para su uso con los datos seleccionados.	8d	15/10/19	23/10/19
28	2.3. Iteración 3		31d	22/10/19	26/11/19
29	Informe de experimentación numérica	Se realizan las pruebas con el algoritmo y el simulador y se obtienen los reportes de experimentación numérica.	3d	22/10/19	24/10/19
30	Diseño del prototipo para visualizar el funcionamiento del algoritmo Cuckoo Search	Se hace una evaluación del contenido que tendrá el videojuego.	4d	25/10/19	29/10/19
31	Desarrollo del prototipo	Teniendo el diseño, se procede a desarrollar el prototipo con las herramientas definidas. Incluyendo también, el documento de diseño de juego.	15d	30/10/19	15/11/19
32	Pruebas funcionales del prototipo	Se realizan las pruebas del funcionamiento del videojuego, evaluando si el algoritmo se ejecuta dentro de éste.	3d	16/11/19	19/11/19
33	Correcciones finales	Se hace una revisión de todo el proyecto y se realizan las correcciones correspondientes en base a las observaciones que se han tenido durante el desarrollo.	6d	20/11/19	26/11/19

Figura A2: Lista de tareas

8. Cronograma del proyecto

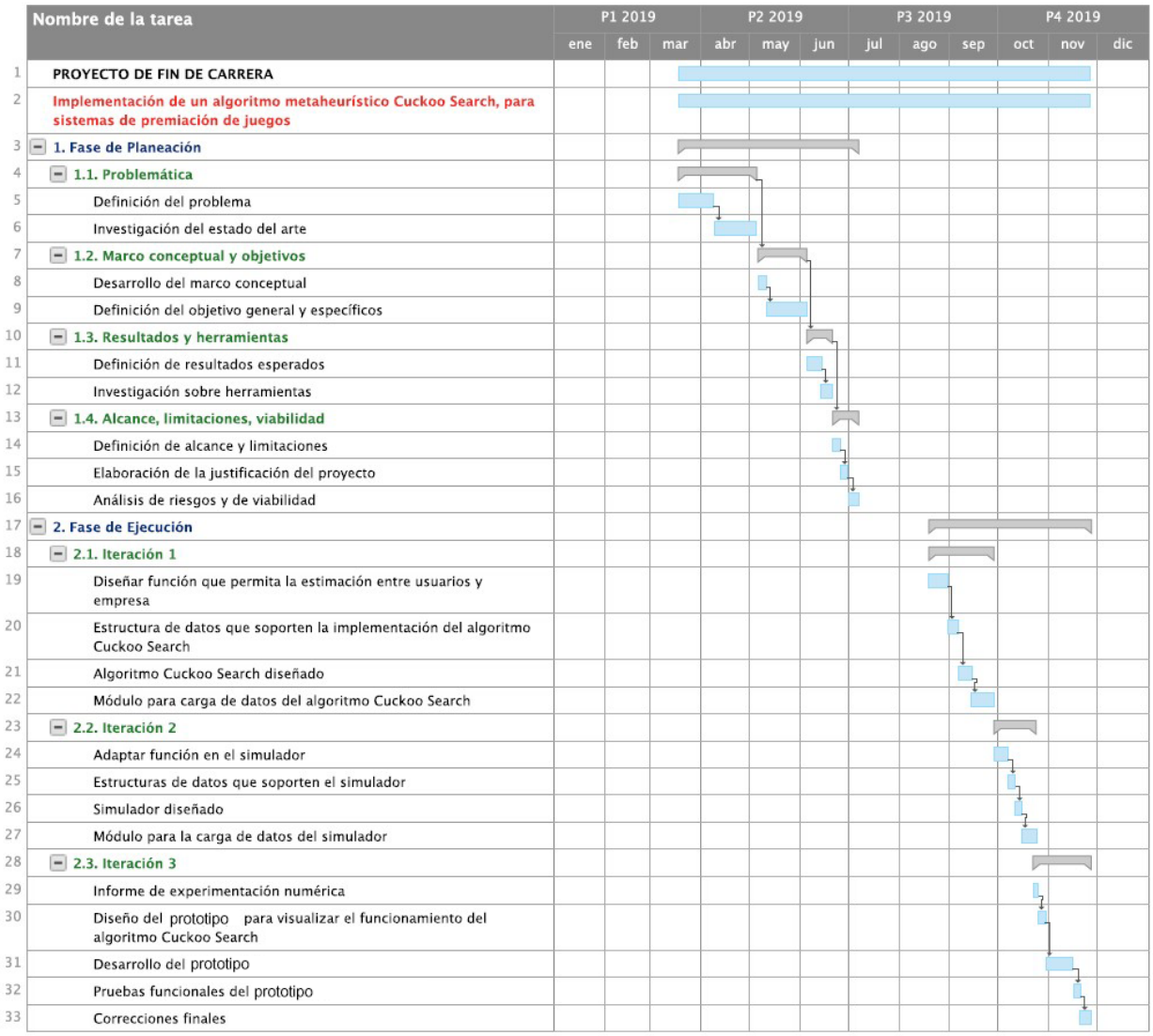


Figura A3: Cronograma Gantt

9. Recursos y costeo

Tabla A2: Tabla de Recursos y costeo

ítem	Descripción	Unidad	Cantidad	Valor Unidad (S/.)	Monto Parcial (S/.)	Monto Total (S/.)
1	PSeInt	Unidad	1	0	0	0
2	Visual Studio IDE	Unidad	1	0	0	0
3	Unity Personal	Unidad	1	0	0	0
4	Android Studio	Unidad	1	0	0	0

Anexo B

1. Archivo Prueba Resultados

Parte 1:

pesoPremio	incurrencyGastada
5	10
7	20
4	30
6	40
3	50
1	60
6	70
5	80
2	90
4	100
4	10
2	20
2	30
1	40
2	50
8	10
7	20
9	30
5	40
4	50
5	60
7	70
3	80
4	90
6	100
3	110
6	120
9	130

montoInvertido	pesoInversion
100	10
300	10
500	10
200	10

cantDiasSeguidos	pesoDias
40	6

diasSeguidos	pesoDias
[1;7]	1
[8;30]	2
[31;180]	6
[180;+∞[	10

monto	pesoInversion
[0;49]	2
[50;99]	5
[100;+∞[	10

Usuario	Empresa	F.O
821450	456560	0.79921588

Figura B1: Archivo Entrada Usuario (Historial de *rolleos*)

Parte 2:

N° Rolls	Veces Invertidas	TotalIncurrencyGastada	TotalInvertido
303	4	3030	1100

incurrencyxMonto	incurrencyxMes	TotalIncurrencyAcumulada	pesoIncurrency
10	100	11133	40

incurrencyAcumulada	pesoIncurrency
[0;100]	5
[101;1000]	10
[1001;5000]	20
[5001;+∞]	40

Figura B2: Archivo Entrada Usuario (Resmen Usuario)

Leyenda:

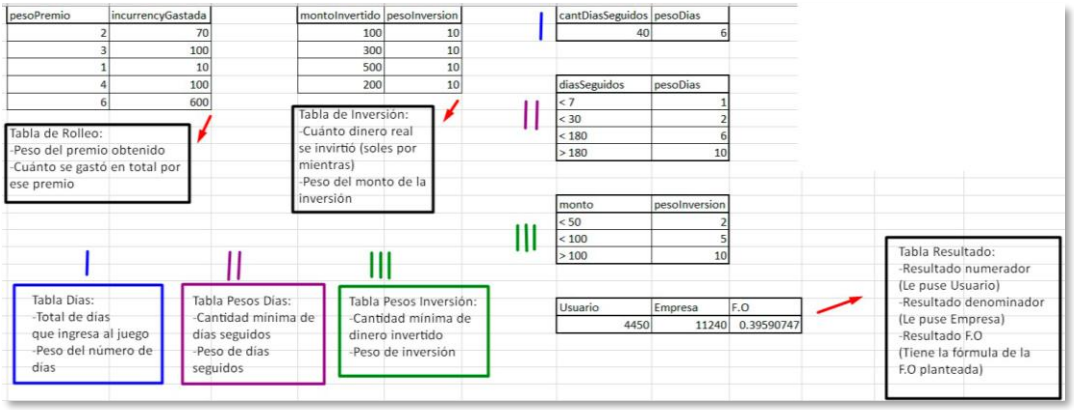


Figura B3: Leyenda Figura B1

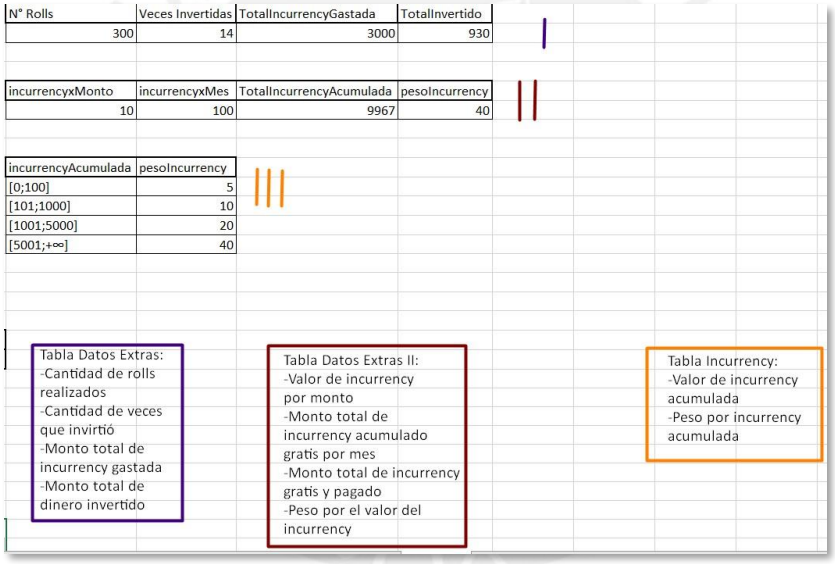


Figura B4: Leyenda Figura B2

2. Archivo Resultado Cuckoo Search vs Simulador

Usuario que invierte

CS				Sim			
Usuario	830830	Empresa	472160	Usuario	825630	Empresa	472160
Roll	Nombre	Peso	Fitness	Roll	Nombre	Peso	Fitness
1	Be Elegant	3	0.797706583	1	Euryale	1	0.797662815
2	Gandr	3	0.794630374	2	Cu Chulainn	1	0.794542992
3	Romulus	1	0.791586706	3	The Blue Black Keys	1	0.791455862
4	Ley Line	1	0.78853187	4	The Green Black Keys	1	0.788401254
5	Lancelot	7	0.785639777	5	Azoth Blade	1	0.785378999
6	Grimoire	1	0.783169906	6	Ley Line	1	0.782388927
7	Darius III	1	0.780210499	7	Azoth Blade	1	0.779430873
8	Azoth Blade	1	0.777282947	8	Darius III	1	0.77650467
9	Gilles de Rais	1	0.774387086	9	Gilles de Rais	1	0.773610152
10	Ley Line	1	0.771522751	10	Be Elegant	3	0.770790245
11	Imaginary Number Magecraft	3	0.768861838	11	Moony Jewel	4	0.768410186
12	Attila	9	0.766660941	12	Darius III	1	0.766253006
13	Gilgamesh	10	0.765967936	13	Azoth Blade	1	0.763481653
14	Siegfried	7	0.765662444	14	Medusa	1	0.760741185
15	Nero Claudius	8	0.764759911	15	Magic Crystal	1	0.758031442
16	Imaginary Number Magecraft	3	0.764201638	16	Moony Jewel	4	0.75541624
17	Be Elegant	3	0.762538317	17	Kiyohime	1	0.753725307
18	Dragonkind	1	0.761603196	18	Nero Claudius	8	0.751253825
19	Artoria Pendragon	9	0.759164969	19	Euryale	1	0.751188052
20	Bath of the Lunar Goddess	6	0.75974246	20	Azoth Blade	1	0.748623348

Figura B5: Archivo Usuario P2W

Usuario Free to Play:

CS				Sim			
Usuario	69140	Empresa	69140	Usuario	66060	Empresa	69140
Roll	Nombre	Peso	Fitness	Roll	Nombre	Peso	Fitness
1	Steel Training	3	0.012187195	1	The Blue Black Keys	1	0.011862203
2	Gandr	3	0.006134969	2	Mooncell Automaton	2	0.00565063
3	Angel's Song	3	0.000641643	3	The Green Black Keys	1	0.000320821
4	Jeanne d'Arc	9	0.003347147	4	Gaius Julius Caesar	1	0.00621613
5	Kaleidoscope	5	0.004592968	5	The Red Black Keys	1	0.011878366
6	Rin's Pendant	1	0.007554297	6	Ushiwakamaru	1	0.017311929
7	Grimoire	1	0.012824523	7	Mooncell Automaton	2	0.022364717
8	Euryale	1	0.017873795	8	Mooncell Automaton	2	0.026266708
9	Magic Crystal	1	0.022706209	9	Elizabeth Bathory	7	0.029039234
10	Divine Banquet	3	0.027018729	10	Gaius Julius Caesar	1	0.026251151
11	Tamamo Cat	9	0.027464144	11	Alexander	1	0.030668294
12	Verdant Sound of Destruction	3	0.019259933	12	Magic Crystal	1	0.034880194
13	The Blue Black Keys	1	0.020048236	13	Gaius Julius Caesar	1	0.038890564
14	Lancelot	7	0.023074618	14	Dragonkind	1	0.042703027
15	Steel Training	3	0.014894251	15	Medea	1	0.04632112
16	The Green Black Keys	1	0.008143322	16	Heaven's Feel	5	0.049156056
17	Artoria Pendragon	8	0.010597586	17	Mooncell Automaton	2	0.043420665
18	Gilles de Rais	2	0.002633889	18	Mooncell Automaton	2	0.044044483
19	Gilles de Rais	1	0.001891184	19	Mooncell Automaton	2	0.04437009
20	Kaleidoscope	5	0.000578536	20	Medusa	1	0.044547295

Figura B6: Archivo Usuario F2P

3. Game Design Document (GDD)

Tabla B1: Tabla resumen GDD

TÍTULO	PXCP WARRIORS
GÉNERO	MMORPG, táctico por turnos
AUDIENCIA	Jugadores casuales, personas mayores a 12 años, mercado occidental
PLATAFORMAS	Móviles (Smartphones Android)
MODOS DE JUEGO	Campaña (historia) single player, modo multiplayer
TEMÁTICA	Parodia
ESTÉTICA	Dibujos

Descripción del Juego:

Como estudiante recién ingresado en la gran universidad PXPC, tendrás que adaptarte a este nuevo entorno para poder egresar y conseguir tu título mientras logres acabar todos tus cursos.

Personajes:

- **Protagonista:** Personaje. Elije las opciones disponibles que aparecen en pantalla para poder avanzar en la historia, comprar en la tienda, reclutar algún personaje, ver el historial de reclutamientos hechos o configurar el juego. Tiene como inventario monedas PXCP que le sirven para poder reclutar algún personaje disponible en el gacha.
- **Reclutas:** NPC. Realizan las acciones correspondientes a lo que el protagonista les indica como atacar o defender. Cada recluta tiene un tipo (Stone, Scissor o Paper) y nivel (5, 4 o 3).
- **Equipo:** NPC. Objetos que le sirve a los reclutas para darles nuevas habilidades o mejorar sus características. Cada equipo tiene un nivel (5, 4 o 3).

Mecánicas de juego:

- **Escoger opciones:** Según esté en la pantalla, se visualizarán botones para que el protagonista elija.

- **Tienda:** Dependiendo de la opción escogida, puede recargar sus monedas PXCP en 100, 500 o 1000 (se utiliza dinero real).
- **Reclutar:** Gastando un monto de 10 monedas PXCP, el protagonista puede reclutar personajes y cartas equipo que le ayuden a superar las misiones u obstáculos del modo historia.
- **Historial:** El protagonista puede revisar su historial de reclutamientos a lo largo del juego, desplazando la pantalla hacia arriba o abajo. Dichos reclutas están organizados desde el más antiguo al más reciente.

Controles:

Todas las acciones que puede realizar el protagonista se hacen mediante la presión de la pantalla, las opciones que aparezcan activarán su funcionamiento si los toca una vez. Para comprobar que está tocando la pantalla, se visualiza un pequeño destello en el lugar donde se presionó.

Audio:

- **Pantalla de inicio:** Música de fondo de tipo aventura.
- **Pantalla de menú:** Música de fondo de tipo misterio.
- **Pantalla de reclutar:** Música de fondo de tipo batalla.
- **Pantalla de premio obtenido:** Música de fondo de tipo sorpresa.
- **Pantalla de historial:** Música de fondo de tipo ambiente.
- **Pantalla de tienda:** Música de fondo de tipo bazar.
- **Tocar pantalla:** Al presionar la pantalla, se activa un sonido corto de tipo iluminación en respuesta.

Lista de Reclutas y Equipos:

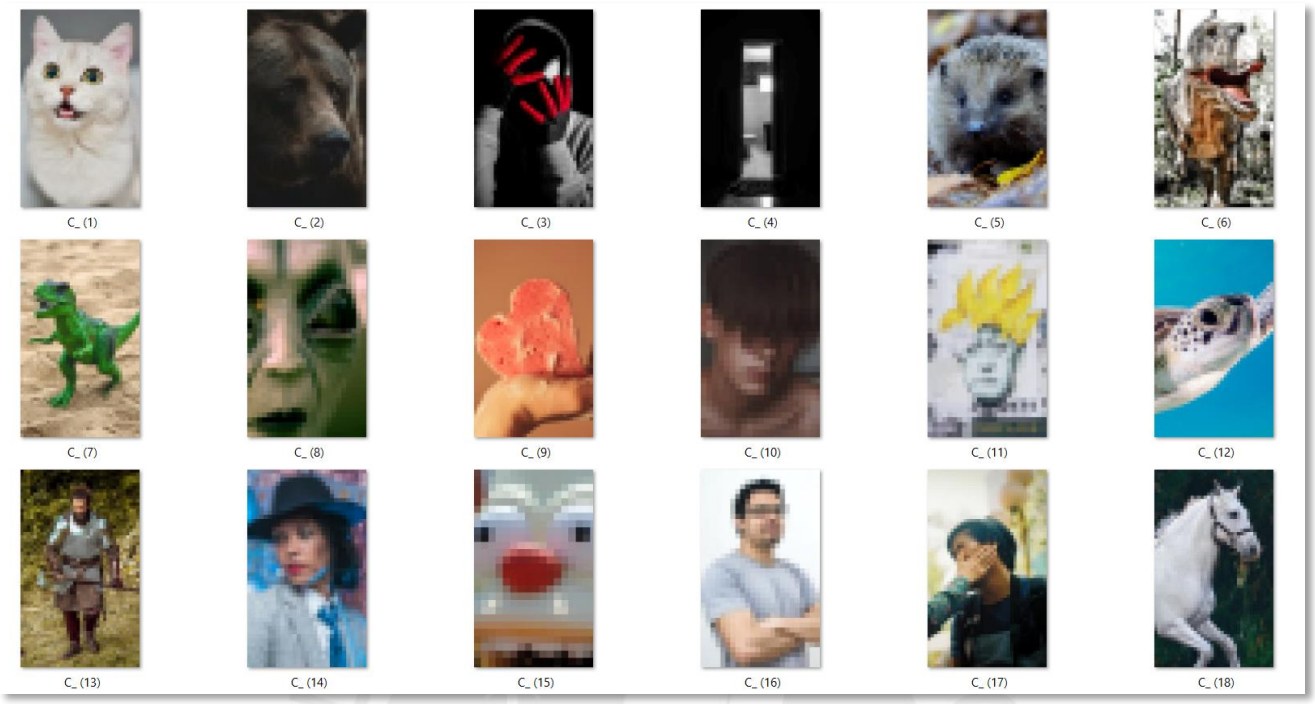


Figura B7: Lista de reclutas y equipos 1-18

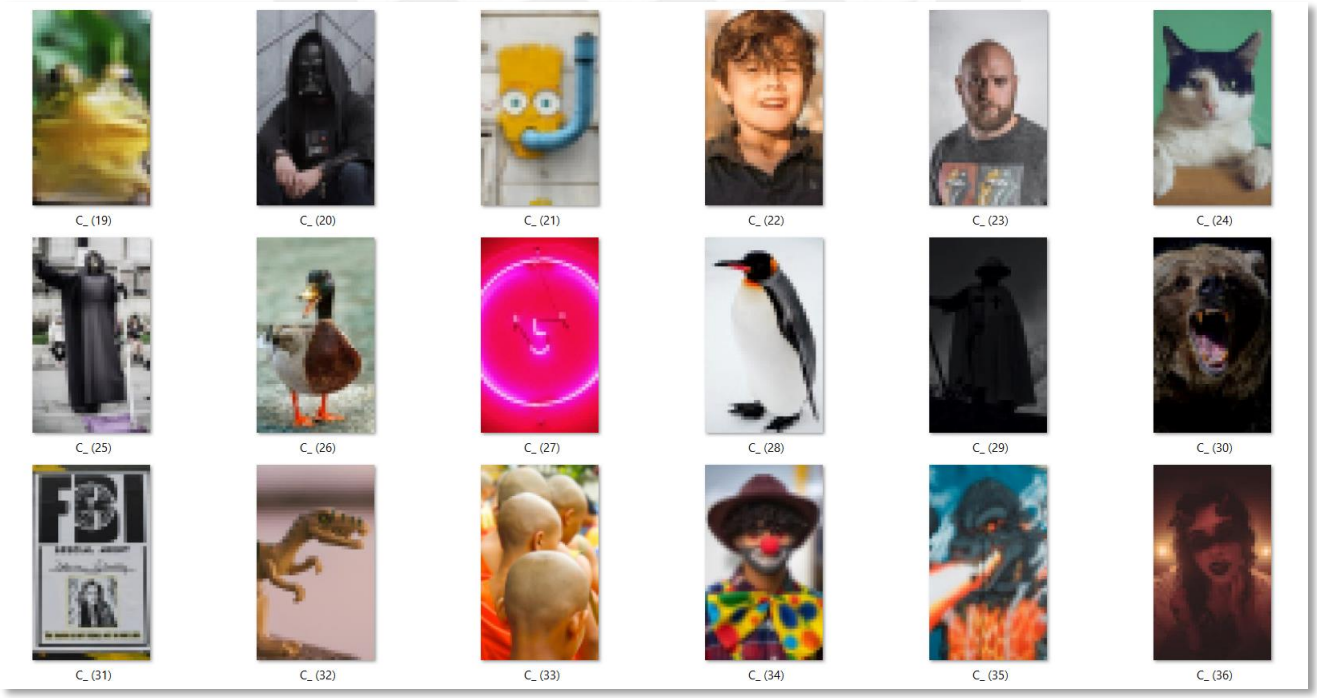


Figura B8: Lista de reclutas y equipos 19-36

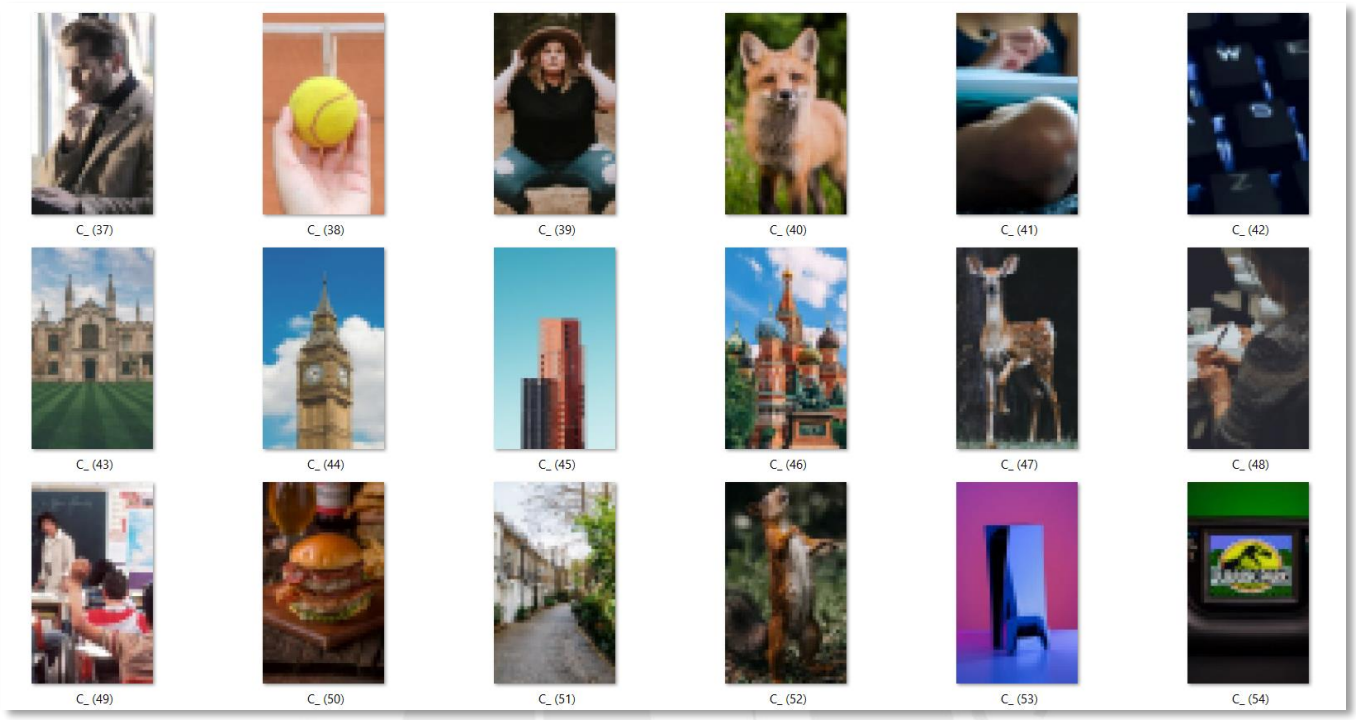


Figura B9: Lista de reclutas y equipos 37-54

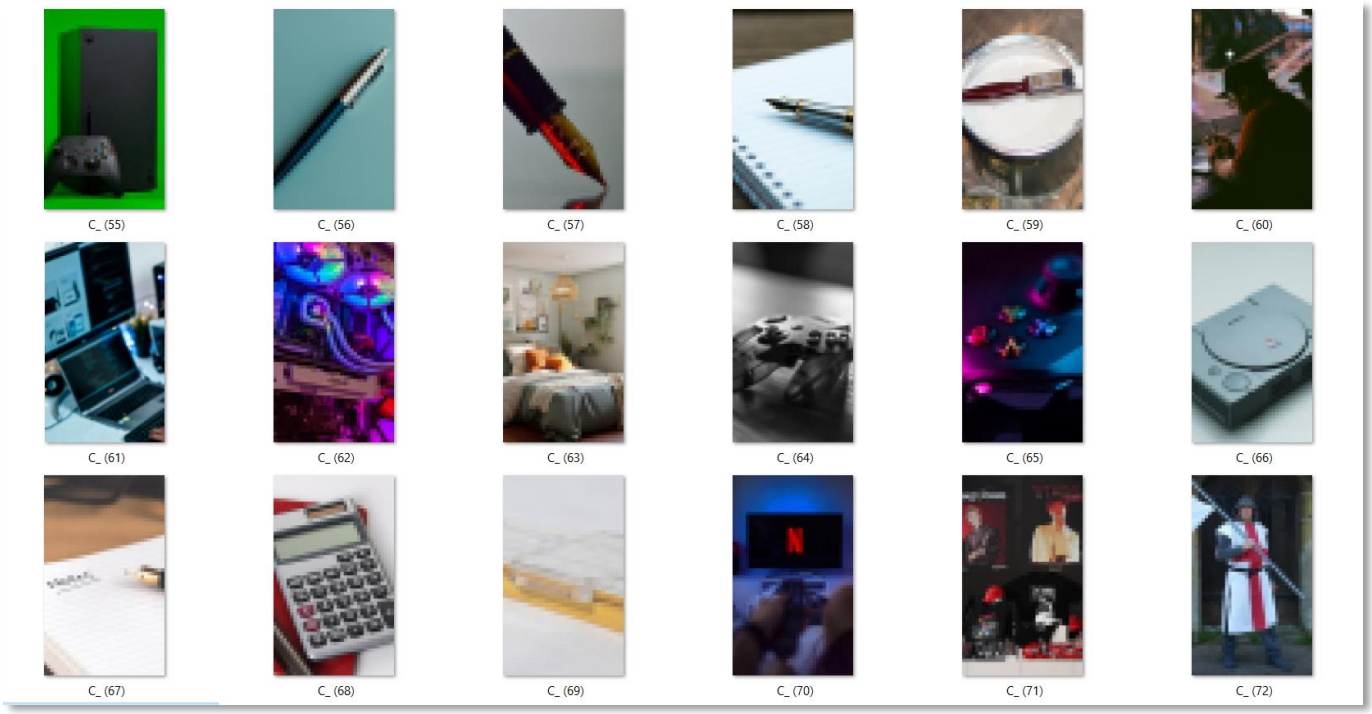


Figura B10: Lista de reclutas y equipos 55-72

4. Pantallas prototipo

Pantalla Título:

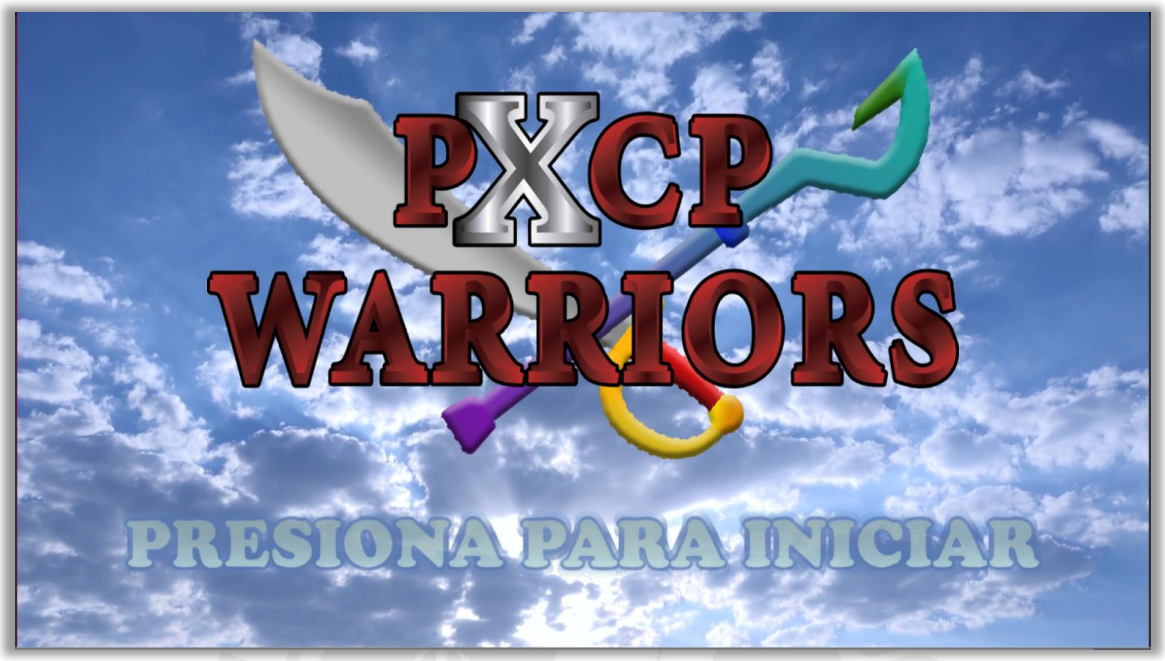


Figura B11: Pantalla título prototipo

Pantalla Menú:

Figura B12: Pantalla menú prototipo

Pantalla Tienda:

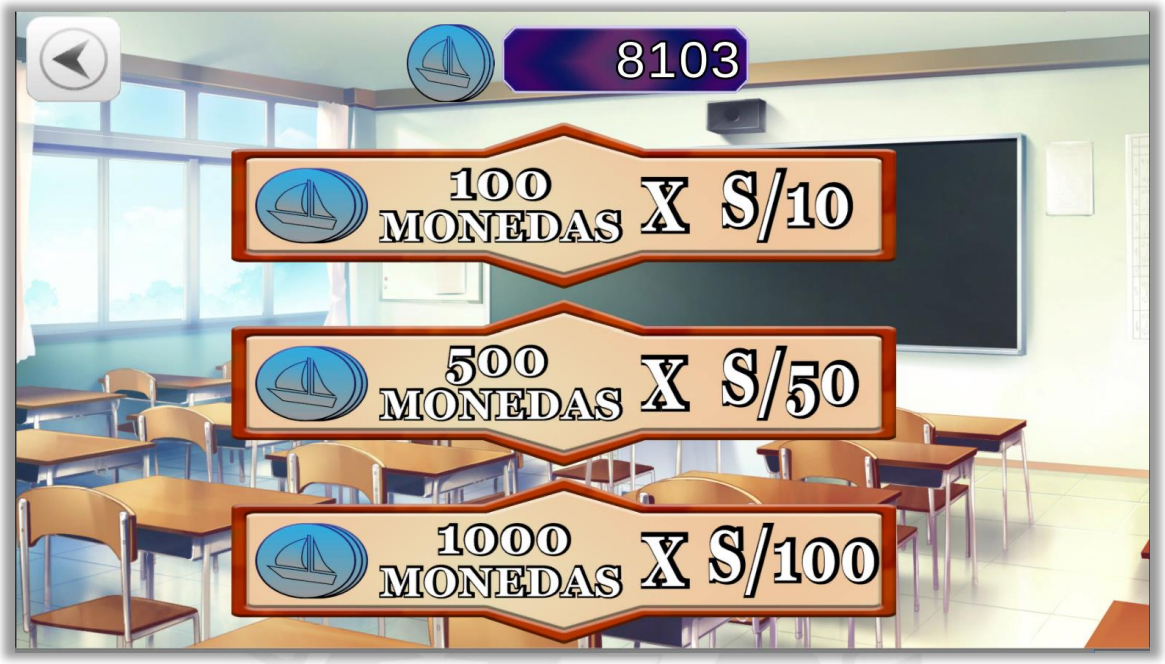


Figura B13: Pantalla tienda prototipo

5. Calibración de variables

Número de Nidos:

Nidos	Iteraciones	pa	
40	1000	0.25	
Resultado			
Usuario	830830	Empresa	472160
1	Be Elegant	3	0.79770658
2	Gandr	3	0.79463037
3	Romulus	1	0.79158671
4	Ley Line	1	0.78853187
5	Lancelot	7	0.78563978
6	Grimoire	1	0.78316991
7	Darius III	1	0.7802105
8	Azoth Blade	1	0.77728295
9	Gilles de Rais	1	0.77438709
10	Ley Line	1	0.77152275
11	Imaginary Number Magecraft	3	0.76886184
12	Attila	9	0.76666094
13	Gilgamesh	10	0.76596794
14	Siegfried	7	0.76566244
15	Nero Claudius	8	0.76475991
16	Imaginary Number Magecraft	3	0.76420164
17	Be Elegant	3	0.76253832
18	Dragonkind	1	0.7616032
19	Artoria Pendragon	9	0.75916497
20	Bath of the Lunar Goddess	6	0.75974246

Figura B14: Calibración número de nidos (40)

Nidos	Iteraciones	pa	
100	1000	0.25	
Resultado			
Usuario	827750	Empresa	472160
1	EMIYA	7	0.79779412
2	Bath of the Lunar Goddess	6	0.79478329
3	False Attendant's Writings	1	0.79180478
4	Verdant Sound of Destruction	3	0.7887931
5	Cu Chulainn	1	0.78590056
6	Steel Training	3	0.78295297
7	Carmilla	7	0.78034044
8	Prisma Cosmos	5	0.77827741
9	Steel Training	3	0.77602728
10	Heracles	7	0.77382799
11	Nero Claudius	8	0.77230299
12	Mooncell Automaton	2	0.77101941
13	Imaginary Around	5	0.76856139
14	Sealing Designation / Enforcer	3	0.76688206
15	Projection	3	0.76475991
16	Gilles de Rais	1	0.76266633
17	Gilles de Rais	1	0.76000511
18	Gandr	3	0.75743795
19	Ley Line	1	0.75555838
20	Parted Sea	1	0.75298628

Figura B15: Calibración número de nidos (100)

Nidos	Iteraciones	pa	
150	1000	0.25	
Resultado			
Usuario1	829360	Empresa	472160
1	Cu Chulainn	1	0.79766282
2	Formal Craft	5	0.79463037
3	Kaleidoscope	5	0.79171755
4	Ley Line	1	0.78883664
5	The Green Black Keys	1	0.78581363
6	Martha	7	0.78295297
7	Be Elegant	3	0.78068694
8	Vlad III	9	0.77819094
9	Stheno	7	0.77663156
10	Artoria Pendragon	8	0.77494829
11	Angel's Song	3	0.7735074
12	Medusa	1	0.77112676
13	Sealing Designation / Enforcer	3	0.76838992
14	Projection	3	0.76619736
15	Azoth Blade	1	0.76403366
16	Lu Bu Fengxian	1	0.76134425
17	Zhuge Liang	9	0.75885559
18	Nero Claudius	8	0.75909555
19	Grimoire	1	0.75901646
20	Gaius Julius Caesar	1	0.7564385

Figura B16: Calibración número de nidos (150)

Nidos	Iteraciones	pa	
200	1000	0.25	
Resultado			
Usuario1	826920	Empresa	472160
1	Holy Shroud of Magdalene	3	0.79770658
2	Lu Bu Fengxian	1	0.79458668
3	Heaven's Feel	5	0.79158671
4	Cu Chulainn	1	0.78870603
5	Romulus	1	0.78568324
6	Kaleidoscope	5	0.78277942
7	Mephistopheles	1	0.78025381
8	Lu Bu Fengxian	1	0.77732618
9	Kaleidoscope	5	0.77464606
10	Cu Chulainn	1	0.77279386
11	Grimoire	1	0.76995871
12	Carmilla	7	0.76728358
13	Nero Claudius	8	0.7660751
14	Azoth Blade	1	0.76512752
15	Be Elegant	3	0.76245301
16	Limited / Zero Over	5	0.76044865
17	The Blue Black Keys	1	0.75906846
18	Medusa	1	0.75643914
19	The Blue Black Keys	1	0.75383995
20	Heracles	7	0.75144019

Figura B17: Calibración número de nidos (200)

Probabilidad pa:

Nidos	Iteraciones	pa	
100	1000	0.4	
Resultado			
Usuario1	828810	Empresa	472160
1	Gaius Julius Caesar	1	0.79766282
2	Mephistopheles	1	0.79454299
3	Gilles de Rais	2	0.79147767
4	Imaginary Number Magecraft	3	0.7885101
5	Elizabeth Bathory	7	0.78574844
6	Angel's Song	3	0.78332176
7	Mephistopheles	1	0.78057866
8	Projection	3	0.7778018
9	Imaginary Number Magecraft	3	0.77525035
10	Mephistopheles	1	0.7730524
11	Attila	9	0.77051789
12	Gilles de Rais	2	0.76945208
13	The Red Black Keys	1	0.76729681
14	Primeval Magic	3	0.76459261
15	Bath of the Lunar Goddess	6	0.76253845
16	The Blue Black Keys	1	0.76134425
17	The Green Black Keys	1	0.75868529
18	Gandr	3	0.75609912
19	Sakata Kintoki	10	0.75441276
20	Ushiwakamaru	1	0.75527364

Figura B18: Calibración nidos (100) pa (0.4)

Nidos	Iteraciones	pa	
150	1000	0.4	
Resultado			
Usuario1	829430	Empresa	472160
1	Azoth Blade	1	0.79766282
2	Artoria Pendragon	9	0.79471776
3	Grimoire	1	0.79180478
4	Martha	7	0.78888018
5	Parted Sea	1	0.78624826
6	Imaginary Number Magecraft	3	0.78330007
7	The Blue Black Keys	1	0.780557
8	Gem Magecraft / Antumbra	3	0.77767209
9	Artoria Pendragon	8	0.77522876
10	The Blue Black Keys	1	0.77356946
11	Formal Craft	5	0.77088352
12	Angel's Song	3	0.76897973
13	Grimoire	1	0.76667524
14	Moony Jewel	4	0.76420746
15	Sakata Kintoki	10	0.76251709
16	Imaginary Number Magecraft	3	0.76255971
17	Medea	1	0.76117592
18	Magic Crystal	1	0.75854301
19	Vlad III	9	0.75610998
20	Gandr	3	0.75662911

Figura B19: Calibración nidos (150) pa (0.4)

Nidos	Iteraciones	pa	
100	1000	0.5	
Resultado			
Usuario1	828780	Empresa	472160
1	Divine Banquet	3	0.79770658
2	Ushiwakamaru	1	0.79458668
3	Bath of the Lunar Goddess	6	0.79160851
4	Jing Ke	1	0.78877133
5	Nero Claudius	8	0.78590056
6	Jeanne d'Arc	9	0.78369056
7	Dragonkind	1	0.7815965
8	Marie Antoinette	7	0.77879626
9	Holy Shroud of Magdalene	3	0.77684738
10	Primeval Magic	3	0.7743666
11	Gandr	3	0.77195888
12	Cu Chulainn	1	0.7695809
13	Jing Ke	1	0.76680384
14	Lu Bu Fengxian	1	0.76425026
15	Nero Claudius	8	0.76168404
16	Gilles de Rais	1	0.76279427
17	Siegfried	7	0.76026056
18	Gem Magecraft / Antumbra	3	0.75971183
19	The Blue Black Keys	1	0.75782841
20	Grimoire	1	0.75525246

Figura B20: Calibración nidos (100) pa (0.5)

Nidos	Iteraciones	pa	
150	1000	0.5	
Resultado			
Usuario1	830170	472160	
1	Gaius Julius Caesar	1	0.79766282
2	Mooncell Automaton	2	0.79456484
3	Gem Magecraft / Antumbra	3	0.79154309
4	Bath of the Lunar Goddess	6	0.78868426
5	The Red Black Keys	1	0.78598748
6	Mooncell Automaton	2	0.78301805
7	Lu Bu Fengxian	1	0.78034044
8	Atalanta	7	0.77754237
9	Euryale	1	0.77555249
10	Artoria Pendragon	8	0.77283695
11	Gilles de Rais	2	0.77137818
12	Sakata Kintoki	10	0.76897973
13	The Green Black Keys	1	0.76832562
14	Be Elegant	3	0.76561965
15	Magic Crystal	1	0.76345694
16	Robin Hood	1	0.76076851
17	Atalanta	7	0.75823825
18	Azoth Blade	1	0.75898929
19	Artoria Pendragon	8	0.75653428
20	Kiyohime	1	0.75815402

Figura B21: Calibración nidos (150) pa (0.5)

Cantidad de generaciones:

Nidos	Iteraciones	pa	
100	500	0.4	
Resultado			
Usuario1	828520	Empresa	472160
1	Vlad III	9	0.79783789
2	Medusa	1	0.79471776
3	Elizabeth Bathory	7	0.79176117
4	Rin's Pendant	1	0.78896726
5	Azoth Blade	1	0.78594402
6	Stheno	7	0.78308313
7	Sakata Kintoki	10	0.78096847
8	Mooncell Automaton	2	0.77922864
9	Mooncell Automaton	2	0.77650207
10	Magic Crystal	1	0.77384953
11	Formal Craft	5	0.77109859
12	Rin's Pendant	1	0.76915149
13	Imaginary Number Magecraft	3	0.76658951
14	Moony Jewel	4	0.76442143
15	Marie Antoinette	7	0.76266661
16	Mooncell Automaton	2	0.76179205
17	Boudica	1	0.76004768
18	Lu Bu Fengxian	1	0.7574167
19	Formal Craft	5	0.75490071
20	Gem Magecraft / Antumbra	3	0.75474415

Figura B22: Calibración nidos (100) pa (0.4) iteraciones (500)

Nidos	Iteraciones	pa	
100	2000	0.4	
Resultado			
Usuario1	834390	Empresa	472160
1	Atalanta	7	0.79779412
2	Gilgamesh	10	0.79487067
3	Steel Training	3	0.79202285
4	Zhuge Liang	9	0.78922849
5	Bath of the Lunar Goddess	6	0.78683501
6	Attila	9	0.78444984
7	Nero Claudius	8	0.78250606
8	Elizabeth Bathory	7	0.78061224
9	Bath of the Lunar Goddess	6	0.77872497
10	The Red Black Keys	1	0.77723199
11	Sealing Designation / Enforcer	3	0.77443221
12	Ushiwakamaru	1	0.77204998
13	Sakata Kintoki	10	0.76946159
14	Ushiwakamaru	1	0.76902174
15	Imaginary Number Magecraft	3	0.76638329
16	Prisma Cosmos	5	0.76437223
17	Steel Training	3	0.76302793
18	Attila	9	0.76213448
19	Sakata Kintoki	10	0.76489308
20	Nero Claudius	8	0.7671552

Figura B23: Calibración nidos (100) pa (0.4) iteraciones (2000)