UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

Año 2021 - 1er Cuatrimestre

Trabajo Profesional en Ingeniería en Informática (75.99)

Propuesta - Juego *roguelike*

79979 – Gonzalez, Juan Manuel (juanmg0511@gmail.com)
92290 – Martins Forgan, Diego (dmforgan89@gmail.com)

Contenido

1. Introducción

El presente documento tiene como objetivo presentar la propuesta de Trabajo Profesional de Ingeniería en Informática de los estudiantes Juan Manuel Gonzalez (padrón 79979) y Diego Martins Forgan (padrón 92290).

El principal campo de conocimiento de nuestra propuesta será la creación procedural de contenido. Procedural Content Generation (PCG[1]) corresponde a la creación de contenido de manera algorítmica con nulas o limitadas *inputs* por parte del usuario. Si bien esta creación de contenido procedural puede ser enfocada a distintas ramas, como pueden ser usos en expresiones artísticas o creación de contenido coherente para algun *bot*, es de nuestro especial interés la aplicación de estas técnicas dentro del ámbito de los videojuegos. Este espacio, además de tener nuestro mayor interés, es uno de los que tiene mayor estudio y aplicación de las técnicas comprendidas en PCG.

En este contexto, serán entonces ejemplos de posible contenido a generar: niveles, mapas, reglas de juego, texturas, items, quests, música, armas, vehículos, personajes, etc.

2. Descripción del Problema

Generalmente para realizar niveles u otros aspectos dentro de un videojuego, es necesario contar con los recursos humanos adecuados y capacitados para tales tareas. Esto implica que detrás de cada decisión de diseño debe haber una o varias personas responsables de dichas decisiones acarreando los costos de tiempo y dinero asociados. Sumado a lo anteriormente mencionado, existe un límite previamente establecido por el equipo de desarrollo en cuanto al contenido desarrollado. Esto significa que por ejemplo un determinado videojuego contará con exactamente una cantidad de niveles que serán iguales cada vez que se juegue.

Con PCG podemos atacar ambos problemas ya que ciertos componentes se crearán algorítmicamente en el momento previo a ser utilizados brindando menor necesidad de recursos humanos para crear estos componentes y además brindar una experiencia similar pero de ninguna manera igual cada vez que se juegue al título.

2.1. Características de un juego *roguelike*

Dentro de los tipos de juegos que utilizan PCG como base, se encuentran los juegos *roguelike*. Una definición simple es que se trata de cualquier juego que haya sido inspirado, en forma directa o indirecta, por el juego *Rogue*[2], lanzado en el año 1980. En forma más específica, se trata de un sub-género de juegos del tipo Role Playing Game (RPG[3]) que comparten ciertas características con dicho juego, como ser la muerte permanente del jugador (conocido en el ambiente como *perma-death*), el uso de PCG para la creación de niveles, enemigos y tesoros, entre otros elementos, y la dinámica de juego por turnos.

[1] Ver [1], sección 1.1.
[2] Ver [3].
[3] Ver [2].

Tomando como premisa esta definición básica, han existido a lo largo de los años diversos intentos por definir el género en forma más estricta. Una de las más populares es la "Interpretación de Berlín"[4], surgida de la *Roguelike Development Conference* del año 2008. En dicho evento, se consensuaron ciertas características *core* que hacen al género, clasificadas como "deseables de alto valor" o "deseables de bajo valor".

Entre las características deseables de alto valor, se incluyen:

- **Generación de entorno en forma aleatoria:** el mundo del juego debe ser generado en forma aleatoria (o procedural), de una forma que incentive la re-jugabilidad.

- **Perma-death:** no se espera que el jugador pueda ganar en su primera partida. El juego recomienza desde el principio al morir (se pueden salvar *check-points*, pero se pierden al finalizar la partida). El juego debe hacer que esto sea disfrutable, en vez de un castigo.

- **Basado en turnos:** el juego debe ser basado en turnos, y cada turno representa una acción.

- **Basado en grilla:** el mundo se divide en una grilla uniforme de cuadrados, o tiles. El jugador y los enemigos ocupan 1 cuadrado, independientemente de su tamaño.

- **No modal:** el movimiento del jugador, las batallas y otras acciones tienen lugar en el mismo modo.

- **Complejo:** el juego debe ser lo suficientemente complejo como para permitir que exista más de una solución a una situación dada.

- **Administración de recursos:** el jugador deberá administrar en forma adecuada recursos limitados (comida, pociones, etc.) para poder finalizar el juego.

- **Hack'n'slash:** si bien el juego puede incluir muchas más cosas, matar muchos monstruos es una parte muy importante de un juego *roguelike*. El juego es "jugador vs. mundo". No existen alianzas, diplomacia u otras ayudas similares.

- **Exploración y descubrimiento:** el juego requiere una exploración minuciosa de los niveles y el descubrimiento (y posterior uso) de diversos elementos.

Entre las características deseables de bajo valor, se incluyen:

- **Un solo jugador:** el jugador controla un solo personaje. El juego es está centrado en el jugador, y el mundo es visto a través de dicho personaje. La muerte del jugador significa el fin de la partida.

[4] Ver [4].

- **Enemigos similares al jugador:** las reglas que aplican a los jugadores, también aplican a los enemigos. Tienen inventarios, equipamiento, pueden usar hechizos, etc.

- **Desafío táctico:** el jugador debe aprender las tácticas del juego antes de poder lograr un progreso importante en el juego. Este proceso se repite a medida que progresa la partida (las tácticas de un nivel más avanzado deben ser distintas a la de un nivel inferior).

- **Gráficos ASCII:** la forma tradicional de representar un juego *roguelike* es a través de una interfaz basada en texto, usando el set de caracteres ASCII.

- **Calabozos:** los juegos *roguelike* contienen calabozos, definidos como niveles compuestos por habitaciones y pasillos.

- **Números:** los números utilizados para describir al personaje (salud, atributos, etc.) se muestran deliberadamente.

Esta definición, si bien es popular y resulta aceptada en el ambiente, ha generado cierta polémica dada algunas de las características, tal como menciona el reconocido desarrollador *roguelike* Darren Grey en su blog[5]. Por ejemplo, podemos citar el uso de caracteres ASCII para la representación gráfica o la necesidad de incluir calabozos.

En base a este tipo de críticas, en la comunidad se ha adoptado esta definición en forma parcial, y el término *roguelike* ha pasado a ser sinónimo de “juego con elementos o características *roguelike*”, en vez de adoptar la definición en forma exacta.

3. Estado del Arte

La creación de contenido procedural forma parte de la industria de los videojuegos hace ya varias décadas. En los últimos años también hubo un aumento en el estudio de PCG por parte de la academia[6]. Esto posibilitó la aparición de nuevos métodos y la mejora de métodos anteriormente utilizados donde incluso algunas de estas mejoras aún requieren de mayor investigación para poder ser implementados en un videojuego comercial.

Sin embargo, existen aún una serie de problemas sin resolver que pueden deberse a limitaciones propias de las tecnologías actuales o que tal vez nunca tengan una solución.

Estos problemas son:

- **Multi-nivel y multi-contenido PCG:** se refiere a un generador de contenido que a partir de un motor de juego y un set de reglas, sería capaz de generar todo el contenido para el juego logrando un producto de alta calidad y donde los componentes se corresponden perfectamente unos con otros.

[5] Ver [5].
[6] Ver [1], apéndice A.

- **Diseño de juego basado en PCG:** consiste en crear juegos donde PCG forme una parte central del gameplay de forma tal que si se quitara el módulo de creación de contenido no quedaría nada reconocible del juego.

- **Generar juegos completos:** se refiere a un generador capaz de generar no sólo contenido para el juego sino el juego en sí. Esto implica las reglas, la estructura de recompensas, la representación gráfica, niveles, personajes, etc.

4. Objetivos

El Trabajo Profesional consta de 4 objetivos principales:

- Diseñar y construir un juego del género RPG.

- Dotar al juego de componentes *roguelike*, investigando e implementando para ello los distintos métodos existentes de generación procedural de contenido (PCG).

- Dotar al juego de una curva de dificultad adaptativa, investigando para ello los distintos métodos existentes para su implementación.

- Analizar los resultados obtenidos como una alternativa viable a la generación de contenido tradicional para videojuegos de tipo RPG.

5. Características del Trabajo

En esta sección presentaremos las principales características del Trabajo Profesional propuesto.

5.1. Diseño general

A continuación son tratados los aspectos generales que se desprenden del diseño del juego a desarrollar.

5.1.1. Juego

Proponemos crear un videojuego completo utilizando métodos de PCG dentro del mismo. Este juego será del género RPG con elementos *roguelike*, esto quiere decir que por ejemplo los distintos calabozos o escenarios sobre los cuales se desplazará el personaje y los enemigos serán generados de forma procedural. Cada vez que el jugador comienza una nueva partida, tendrá una experiencia distinta.

Adicionalmente, estos niveles deben cumplir con los requisitos del juego, es decir que el jugador pueda atravesarlo (que a partir de un punto de inicio, sea posible llegar hasta el final del nivel) y que la ubicación de enemigos y objetos sea lógica.

Los detalles del juego, como ser el nombre, los objetivos, y el reglamento serán diseñados durante el desarrollo del proyecto.

5.1.2. Gráficos y presentación

El juego será implementado como una aplicación 2D, con una visualización *top-down* o *birds-eye* del mundo, utilizando sprites NO generados por nosotros ya que entendemos que el apartado gráfico no es el foco del trabajo profesional. Todo el contenido de este tipo será de uso libre. Los assets que puedan ser realizados por nosotros, serán aclarados e informados.

5.1.3. Efectos de sonido y música

En forma similar, utilizaremos efectos de sonido y música o bien de uso libre o generados por nosotros en forma *online*.

5.2. Generación de contenido en forma procedural

Planteamos como objetivo la implementación de 3 métodos, como mínimo, para la generación de los escenarios, dando la posibilidad al usuario el cambio entre un método u otro en forma de configuración o *setting,* antes de iniciar la partida. Por otro lado, también contemplaremos la implementación de al menos un método de PCG para la ubicación de los distintos elementos en el mapa, como por ejemplo enemigos, items, etc.

Dado que existe una gran variedad de estrategias para lograr este objetivo, a continuación citamos algunos ejemplos con los algoritmos más populares:

- Iteración básica.
- Árbol BSP.
- Autómata celular.
- Búsqueda DFS.

5.3. Dificultad adaptativa

En pocas palabras, esta funcionalidad apunta a dotar a los distintos enemigos con una IA competente para que el juego sea “jugable” y mínimamente desafiante.

Para ello, diseñaremos el juego con una curva de dificultad adaptativa (DDA[7]), basada por ejemplo en los conceptos desarrollados para el sistema de DDA Hamlet[8], es decir que si el juego es superado con facilidad por el usuario, cada nivel generado de manera procedural aumentará su dificultad en función de ciertos parámetros y así también lo harán los enemigos. Para evitar que el juego sea frustrante, sucederá lo contrario si el usuario comienza a perder muchas veces.

[7] Ver [6].
[8] Ver [7].

5.4. Uso de la nube

Finalmente, se contará con la posibilidad de guardar los puntajes en la nube, sincronizándose en los distintos dispositivos del usuario, y generando ránkings de jugadores.

6. Tecnologías

En esta sección discutiremos las principales tecnologías que formarán parte de este Trabajo Profesional. Como premisa, el juego se desarrollará para la plataforma de PC debido a que es la que a nuestro criterio cuenta con menores limitaciones de performance a comparación de otras, como por ejemplo mobile.

6.1. Unity[9]

Para la implementación del juego, utilizaremos el motor Unity[10], que actualmente representa no sólo el estado del arte en creación de videojuegos, si no que se trata además de uno de los más populares del mercado.

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, macOS y Linux.

El éxito de Unity ha llegado en parte debido al enfoque en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles. El enfoque de la compañía es "democratizar el desarrollo de juegos", y hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a tantas personas en todo el mundo como sea posible.

El motor gráfico utiliza OpenGL (en Windows, Mac y Linux), Direct3D (solo en Windows), OpenGL ES (en Android y iOS), e interfaces propietarias (Wii). El scripting se basa en Mono, la implementación de código abierto de .NET Framework. Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), C# o Boo (que tiene una sintaxis inspirada en Python).

6.2. CloudSync

A fin de implementar la funcionalidad descripta en la sección 5.4, se desplegará un servicio de sincronización de puntaje en la nube, realizado en forma *custom*.

[9] Adaptado de [8].
[10] Ver [9].

El mismo estará provisto de una API REST, que permitirá la interacción con la aplicación. La implementación del server será utilizando Python sobre Gunicorn, montado en Docker[11]. El hosting está a cargo de Heroku[12].

Si bien para el Trabajo Profesional este servicio se desplegará utilizando los *free tiers* de estas aplicaciones, se contemplará y diseñará la infraestructura para una fácil migración a un uso productivo.

6.3. Herramientas y otras tecnologías

Presentamos en la siguiente tabla las herramientas y tecnologías que utilizaremos para la concreción del trabajo:

Categoría	Herramientas
Lenguajes de programación	C# Python Unity Scripting API
Entorno de desarrollo	MS Visual Studio Unity Development Platform
Implementación de servers	Docker Kubernetes
Hosting de servicios en la nube	Heroku
Control de versiones	Git
Administración y gestión del proyecto	Github Projects

7. Alcance

El alcance del trabajo profesional está conformado por los siguientes puntos:

- Desarrollo del juego de tipo RPG con elementos *roguelike*.
    - Diseño de un juego completo del tipo RPG.
    - Implementación de al menos 3 estrategias de generación de niveles, en forma procedural (PCG).
    - Implementación de al menos 1 estrategia de generación de contenido, en forma procedural (PCG).
    - Implementación de una curva de dificultad adaptativa tanto en la generación de niveles como para los personajes NPC.

[11] Ver [10].
[12] Ver [11].

- Desarrollo y despliegue de un servicio de en la nube para la publicación y consulta de *leaderboards* y *high scores*.

8. Plan de Trabajo

En esta sección del documento presentaremos el plan de trabajo ideado para lograr los objetivos propuestos.

8.1. Equipo de Trabajo

El equipo de trabajo estará compuesto por:

Tutor
Leandro Ferrigno

Estudiantes/Desarrolladores
Juan Manuel Gonzalez
Diego Martins Forgan

8.2. Metodología

Como metodología principal de trabajo, basaremos nuestra estrategia en un fuerte trabajo de investigación, centrado en la bibliografía, publicaciones científicas y fuentes online citadas en la sección 9, para luego aplicarlo al diseño y construcción del juego.

En cuanto al desarrollo de la aplicación, trabajaremos haciendo uso de un proceso ágil, basado en SCRUM, definiendo una serie de iteraciones que pueden ser consultadas en la sección 8.3. Cada una de ellas tendrá asociado un entregable, que se presentará en una reunión de avance, a realizarse en una fecha a definir con nuestro tutor.

Adicionalmente, durante cada una de las reuniones de avance, será presentada la planificación detallada de las tareas, objetivos, prioridades, fecha de finalización y entregables de la iteración siguiente. En base a la presentación realizada y el *feedback* de nuestro tutor, serán realizados los ajustes necesarios.

8.3. Estimación de esfuerzo

En esta sección mostramos el listado de iteraciones que hemos definido, junto con las tareas *macro* necesarias para la concreción de la propuesta. Adicionalmente, se incluyen otras etapas y actividades que son relevantes al proyecto.

Cada tarea lleva asociada una estimación en horas, junto con la cantidad de recursos asignados, que puede en este caso ser de 1 o 2 desarrolladores.

Iteración	Descripción	Recursos	Esfuerzo	Total
-	*Pre-Propuestas*	2	5	10
	Investigación previa	2	5	10
	Propuesta	2	15	30
1	**Inicial: Juego RPG**			
	Diseño general y reglas[13]	2	5	10
	Diseño funcional[14]	2	5	10
	Armado de ambientes	1	10	10
	Construcción de aplicación básica en Unity para tomar de base.	1	20	20
	Documentación inicial	2	5	10
2	**PCG: Generación de niveles/enemigos**			
	Investigación	2	15	30
	Diseño	2	25	50
	Construcción	2	45	90
	Testing/Rework	2	10	20
	Documentación	2	10	20
3	**DDA: Dificultad Adaptativa**			
	Investigación	2	10	20
	Diseño	2	10	20
	Construcción	2	30	60
	Testing/Rework	2	5	10
	Documentación	2	5	10
4	**Integración: Juego RPG**			
	Integración de PCG y DDA	2	15	30
	Testing/Rework	2	5	10
5	**UI y Ajustes: Juego RPG**			
	Aprovisionamiento/generación de Assets: sprites, fonts, sfx, música	2	5	10
	Construcción de UI y Ajustes	1	40	40
	Testing/Rework	2	5	10
	Documentación	2	5	10
6	**Construcción: CloudSync**			
	Armado de ambientes	1	5	5
	Diseño API REST	1	10	10
	Construcción	1	20	20
	Testing/Rework	2	5	10

[13] Esta tarea comprende el diseño del juego en sí: nombre, objetivo, mecánica, reglas, etc.
[14] El objetivo de esta tarea es diseñar los aspectos de funcionamiento de la aplicación en sí, como ser: tipo de pantallas, menús, forma de controlar al personaje, etc.

Iteración	Descripción	Recursos	Esfuerzo	Total
6	*Documentación*	1	5	5
7	**Integración: Juego RPG y CloudSync** *Deploy CloudSync en Internet* *Configuración CloudSync en Internet* *Testing/Rework solución completa* *Documentación definitiva*	 1 1 2 2	 3 2 20 10	 3 2 40 20
-	*Reuniones*	2	10	20
-	*Presentación*	2	15	30

Junto con la estimación precedente, debemos contabilizar un 10% adicional al total de horas presentado, en concepto de administración/gestión de proyecto. En consecuencia, la estimación final en horas, en forma consolidada, resulta:

Descripción	Esfuerzo
Iteración 1 *Inicial: Juego RPG*	60
Iteración 2 *PCG: Generación de niveles/enemigos*	210
Iteración 3 *DDA: Dificultad Adaptativa*	120
Iteración 4 *Integración: Juego RPG*	40
Iteración 5 UI y Ajustes*: Juego RPG*	70
Iteración 6 *Construcción: CloudSync*	50
Iteración 7 *Integración: Juego RPG y CloudSync*	65
Otros	100
Gestión de proyecto	72
Total	**787**

9. Referencias

Presentamos a continuación las fuentes consultadas para la elaboración de la presente propuesta:

[1]. Noor Shaker, Julian Togelius, and Mark J. Nelson. Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer [2016].

[2]. Role-playing Game. Wikipedia: https://en.wikipedia.org/wiki/Role-playing_game.

[3]. Rogue (video game). Wikipedia: https://en.wikipedia.org/wiki/Rogue_(video_game).

[4]. Berlin Interpretation. Roguebasin: http://roguebasin.roguelikedevelopment.org/index.php?title=Berlin_Interpretation.

[5]. Screw the Berlin Interpretation! Games of Grey. Blog personal de Darren Grey: http://www.gamesofgrey.com/blog/?p=403.

[6]. Mohammad Zohaib, Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review, *Advances in Human-Computer Interaction*, vol. 2018, Article ID 5681652, 12pages, 2018. https://doi.org/10.1155/2018/5681652.

[7]. Hunicke, Robin & Chapman, Vernell. AI for dynamic difficulty adjustment in games. Challenges in game artificial intelligence AAAI workshop 2. 2004. https://www.researchgate.net/publication/228889029_AI_for_dynamic_difficulty_adjustment_in_games.

[8]. Unity (motor de videojuego). Wikipedia: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego).

[9]. Unity: https://unity.com/.

[10]. Docker: https://www.docker.com.

[11]. Heroku: https://www.heroku.com.

A1. Pre-propuesta juego *roguelike*

A continuación incluímos la pre-propuesta original del juego roguelike.

A1.1. Introducción

Los *roguelike* son un subgénero de los juegos de rol que se caracterizan por atravesar distintos niveles que son generados de manera procedural, a menudo basados en una temática de fantasía.
Por lo general en este tipo de juegos, cada vez que el jugador muere finaliza su partida y debe comenzar todo el juego de nuevo (con algunos items mantenidos de su jugada anterior) con una serie nueva de niveles generados de manera procedural.
Son ejemplos de este género los siguientes juegos: Spelunky, The Binding of Isaac, Hades, Crypt of the NecroDancer y Dead Cells.

A1.2. Propuesta

- Desarrollar un juego *roguelike* en 2D, en forma de aplicación mobile, que implemente **algoritmos de generación procedural** de niveles. Estos niveles deben cumplir con los requisitos del juego, es decir que el jugador pueda atravesarlo (que a partir de un punto de inicio, sea posible llegar hasta el final del nivel) y que la ubicación de enemigos y objetos sea lógica.

- Dotar a los enemigos de una **IA lo suficientemente compleja** para que el juego sea un desafío para el jugador del videojuego.

- Para los apartados anteriores puede desarrollarse una **curva de dificultad adaptativa**, es decir que si el juego es superado con facilidad por el usuario, cada nivel generado de manera procedural aumentará su dificultad en función de ciertos parámetros y así también lo harán los enemigos. Para evitar que el juego sea frustrante, sucederá lo contrario si el usuario comienza a perder muchas veces.

- Utilizar sprites NO generados por nosotros ya que entendemos que el apartado gráfico no es el foco del trabajo final. Los assets que puedan ser realizados por nosotros, serán aclarados e informados.

- *Posible agregado:* Comunicación con los servicios en la nube (propio o de terceros) para poder generar rankings de jugadores y poder transportar el avance del juego entre plataformas.

A2. Pre-propuesta programa de incentivo

A continuación incluímos la pre-propuesta original de la aplicación del programa de incentivo y recompensa para empleados.

A2.1. Introducción

Muchas veces en las empresas surge la necesidad de motivar a los empleados a cumplir con ciertos requerimientos de la corporación, ya sea por razones de formación o regulatorias. Ejemplos de esto pueden ser exámenes de salud, capacitaciones, presentismo o performance de manejo entre otros.

Dado este contexto, se propone un programa que incentive la participación en estas actividades mediante el otorgamiento de puntos, según un reglamento a definir en cada caso, que el empleado podrá canjear por premios mediante el uso de una aplicación en su celular.

A2.2. Propuesta

- Desarrollar una solución que permita implementar el programa de incentivo y recompensa en potenciales clientes, con los siguientes componentes: cliente mobile (Android+iOS), sitio de administración y capa de servicios en la nube.

- El cliente mobile será desarrollado en una tecnología multi-plataforma, como React Native. El empleado podrá ver los puntos obtenidos hasta la fecha, el reglamento vigente, el historial de canjes realizados, el catálogo de premios y asimismo contar con una “tienda” para seleccionar y canjear premios con los puntos disponibles. Adicionalmente, la aplicación enviará notificaciones (de acuerdo a las configuraciones de cada caso) para motivar o recordar sobre el cumplimiento de ciertas actividades relacionadas al programa.

- El conjunto de reglas a aplicar, así como también el formato de los archivos de entrada (con los datos en base a los cuales se calcularán los puntos), será enteramente customizable para cada caso, desde la administración de la solución.

- El sitio de administración será desarrollado en una tecnología moderna, como por ejemplo React, y permitirá definir las reglas del programa, la gestión de la suba de archivos (automática/manual), cuentas de usuarios, canjes y catálogo de premios. Asimismo, permitirá visualizar estadísticas clave del uso de la solución y sus recursos.

- La capa de servicios se basará en servidores que provean una API REST, desarrollados sobre una tecnología .NET Core o similar. Se podrán configurar diversos proveedores de autenticación para los usuarios.

- La capa de servicios en la nube, y el sitio de administración, se implementarán mediante el uso de Kubernetes u otra estrategia similar, que permita escalar dinámicamente de acuerdo a las necesidades en un momento dado.

- *Posible agregado:* integración de autenticación y cuentas con servicios de directorio enterprise, como ser Active Directory de Microsoft.