**Descartado**

**Posible**

**Probable**

**Aprobado**

**Idea #1**

> Package en unity que use los Trees para generar mapas "aleatorios" con los assets del proyecto. Se le podría decir valores como la densidad de cada asset, qué bioma se quiere montar (bosque, ciudad, agua, etc), si se quieren hacer caminos para el player y qué anchura, el tamaño del mapa y el tipo de juego (sandbox, fps, dungeon...), que todo afecte y de un resultado inteligente.

Se le daría al jugador la posibilidad de un controller en 1a persona o en 3a (pillar de unity)

El agua generada proceduralmente

- Estructura:

- Utilidad?

1) Mapas sandbox

2) Mapas sencillos con assets del proyecto que permitan hacer un prototipo rápido (combate, etc)

- Competencia? En qué se diferencia?

- Basarse en Minecraft y su generación. Perlin noise, noises, etc

**Idea #2.a**

[**https://medium.com/techking/human-like-playtesting-with-deep-learning-92adafffe921**](https://medium.com/techking/human-like-playtesting-with-deep-learning-92adafffe921)

Idea combina la simulación por ciclos (muy vista en androides que aprenden a hacer X acción tras muchos intentos) con el diseño de nivel.

El usuario le pasaría a la app el personaje y sería capaz de, con un frustrum de visión y estímulos del entorno, navegar por el mapa y obtener información estilo:

- Visibilidad del entorno, si hay zonas demasiado ocluidas por obstáculos, etc

- Colisión por el nivel, si hay zonas irregulares muy bruscas

- Los enemigos tendrían tag y te diría cuál es el tiempo de detección de un enemigo promedio y por zonas. Simularía el cerebro de una persona con su tiempo de reacción, etc

- Tiempo de completar el nivel. Se podría simular una IA humana simple que trate de resolver el nivel.

- Y en general cualquier elemento que pueda resultar interesante para ajustar el nivel

Cada ciclo haría un recorrido distinto y recopilaría información que se acabaría contrastando y te daría puntuaciones de diseño de nivel sobre cada apartado. Se podría comparar entre versiones del mismo nivel para ver puntos fuertes de distintas iteraciones. Esta comparación serviría para darle validez a los datos obtenidos pues los humanos somos capaces de detectar mejor estas situaciones complejas que involucran tantas variables y son tan subjetivas. Por su parte, la máquina puede automatizar el proceso.

Se usaría Unity y un controller.

**Idea #2.b**

Otra idea sería hacer más énfasis en desarrollar la IA que simula una persona, por tal de poder ejecutar esta "simulación" de nivel una infinidad de veces, en lugar de tener que playtestear, el cual es un método arcaico, mucho más lento y difícil de llevar. Esta IA tendría una gran variedad de "tipos de jugador" de modo que simule el target objetivo del juego.

Desarrollar IA que simule jugadores implicaría que si se quiere lanzar al mercado como un paquete de Unity entonces debe cumplir con el pilar de ser genérico. Preguntas:

- ¿Cómo desarrollas una IA que sea capaz de actuar en función de cualquier juego que le proporcione?

- ¿Cómo haces el setup?

- ¿Cómo accede al código de gameplay del juego target?

Está claro que debería de haber un sistema complejo, genérico y a la vez potente que permita al package y al controller poder obtener acceso a la lógica de juego, desde el nivel pasando por información de enemigos hasta el hecho de que es una IA que imita a un jugador y por lo tanto debe actuar como si fuese alguien delante de la pantalla viendo información que una IA no vería (eg la vida actual en el HUD)

La idea de la IA que simula humanos existe dentro de juegos, pero cabe distinguir humano y jugador. Las primera simula el comportamiento de un humano como entidad dentro del juego, mientras que una IA que simule al jugador se pone en la piel de aquel que está jugando, es decir controla al personaje como si fuese tu amigo al lado jugando.

Por si sólo el concepto se antoja potente, esconde una brutal complejidad de toma de decisiones. Una idea sería centrarse en este concepto con el objetivo de hacer muchas simulaciones dentro de un entorno de juego, pero sin enfocarse necesariamente en el ámbito del diseño de nivel, sino en una o varias situaciones comunes (eg fps y el manejo de un arma).

**Idea #2.c**

Este concepto se puede llevar más allá a temas de UX, de simular una IA que adquiera un reconocimiento de los elementos de la pantalla, algo similar a un eye tracking pero programado de una forma suficientemente fiel como para resultar confiable.

**Posible alternativa de contexto**

Hacer alguna de las anteriores, pero en un entorno único pre-acondicionado. Podría ser en unity, pero con un setup muy intrusivo y no se plantearía como paquete.

Esta idea evita complicaciones del setup en múltiples juegos distintos pero a la vez pierde el valor añadido de ser un paquete redistribuible.

**Idea #2 Puntos en Común**

El objetivo de todas las variantes sería programar una IA que simule comportamiento humano para sustituir (o complementar) el proceso de playtesting humano y así recopilar en un tiempo reducido una gran cantidad de información fiable que pueda ser útil como feedback para el juego.

El medio más práctico y que permite entrar en materia de un modo más accesible es Unity y sus paquetes en la Unity Asset Store, ideales para demos técnicas y utilidades de videojuegos concretas. Es además el entorno más fiable y conservador.

En resumen, estas serían las claves:

***Todas las variantes:***

* IA Simula jugador

1. Jugador ajustable

* Entorno común de juego

1. Entorno variable

* Muchos ciclos de ejecución
* Obtiene info del juego
* Contrasta resultados de todas las iteraciones
* Presenta puntuaciones en varios campos
* Compara los resultados con otras versiones del juego
* Unity y sus paquetes
* Estructura de acceso al código fuente del juego necesario

1. El usuario podría proporcionar acceso a las partes del juego en el inspector de la IA

**Funcionalidad de Unity relacionada:**

<https://blogs.unity3d.com/2013/12/18/unity-test-tools-released/>

<https://blogs.unity3d.com/2018/11/02/testing-test-driven-development-with-the-unity-test-runner/>

<https://forum.unity.com/threads/what-options-do-i-have-for-automation-and-unit-testing-in-unity.682720/>

Todas estas funcionalidades lo que permiten es hacer múltiples simulaciones de la aplicación en otros procesos:

<https://youtu.be/r9HdJ8P6GQI>

Existen varios tipos de simulación y todos tienen métricas de modo que eliges variables y su resultado esperado para que el test se pase, no se pase o sea inconcluso.

**Posible combinación**

Para combinar todas las opciones de modo que pueda:

1. Centrarme en un apartado de desarrollo/programación propio
2. Valerme de una herramienta existente para agilizar el proceso
3. Ser un proceso de playtesting automatizado
4. Ser lo más concreto posible en un campo de videojuegos que me interese más, puesto que abarcar múltiples situaciones de juego puede ser una odisea y además no tiene tanta proyección de futuro
5. Exportar la herramienta resultante en un paquete o framework de Unity

… La propuesta principal sería **programar una IA que imite a un jugador real (1)** mientras me hago valer de un **framework de testing automatizado y de un controller en Unity (2)** y centro el **proceso de playtesting automatizado** **(3**) en un entorno de **diseño de nivel (4)** de modo que pueda exportarse como **paquete de Unity (5)**

**Programación de la IA**

Programar la IA de jugador de modo genérico para que acepte inputs de variables e información del entorno de cada juego en el que se pruebe:

* Al centrarme en diseño de nivel, no existe el problema de tener que acceder a los enemigos de modo que se actúe en función de los scripts y lógica de estos.
* La IA se centraría en navegar por el nivel y tomar decisiones en función de eg. qué/cuántos/etc enemigos ve (GameObject), tendría objetivos (monedas etc)
* La IA tendría en cuenta la User Interface. Se podrían triggear eventos (aleatorios?) de pop-ups dentro del nivel, como al recoger un ítem, etc y ver cómo captan la atención de la IA
* El éxito o fracaso (muerte) del combate con enemigos se podría simular internamente, nada de acceder a scripts sino tocarlo por encima con variables de poder de enemigos, número etc
* La IA se comportaría mediante heurísticas de:

1. Miedo ante enemigos / agresividad
2. Cuánto pondera recoger ítems
3. Cuánto explora. Hay jugadores que tratan de buscar la salida del nivel directo y otros que deambulan…
4. etc

En relación al punto de la exploración, sería interesante extrapolarlo y tener en cuenta los tipos de jugador (eg Bartle Types) y programar una variante de jugador de cada tipo.

Al final de la simulación se mostraría qué tan adecuado es el nivel para cada bartle type según si ha cumplido sus objetivos

Es cierto que existen simulaciones de combate, pero son independientes de la lógica del juego más allá de las variables necesarias para que se simule. Al haberse planteado de un modo genérico, es imposible que esta herramienta sea igual de precisa que si hubiese sido desarrollada por una empresa exclusivamente para su juego.

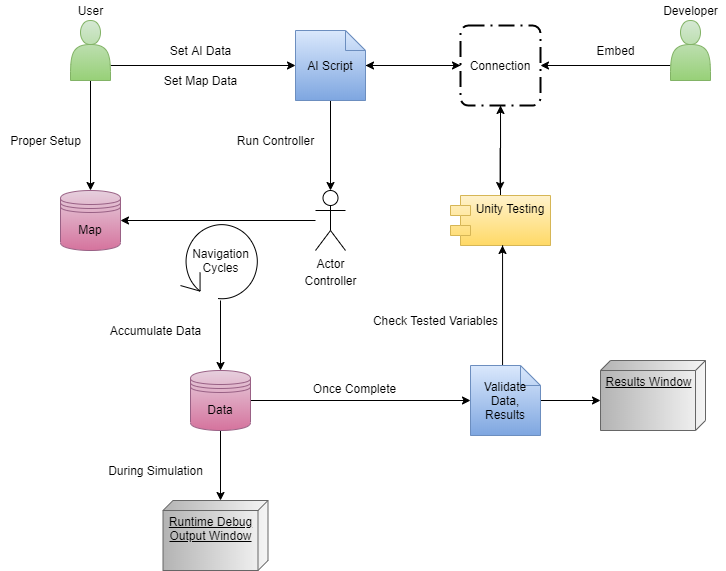
La herramienta además hace un recorrido de principio a fin de nivel y debe atender a muchas variables a lo largo del recorrido, ese es uno de sus atractivos.

**Exportación del paquete**

El/los script(s) de la IA deben de tener ya la lógica del automatizador de Unity con todas las variables ya marcadas.

Sería exportado con la dependencia de este otro paquete de automatización y del controller.

Habría una escena de prueba en el nuevo paquete.

**Diagrama de flow básico**

**Posible añadido → Más allá de la simulación**

Antes de la simulación se haría iteraciones de la IA mediante deep learning. En la simulación estaría disponible elegir una IA con un número de iteraciones concreto, es decir el paquete guardaría un preset del controller con IA iterada.

(Se podría comparar la información obtenida por una IA u otra dentro del mismo setup, quizá incluso realizar una simulación simultánea de dos IAs distintas de modo que se comparen los resultados visualmente)

A la vez, durante las simulaciones, el preset seleccionado acumularía nuevas iteraciones.

Es decir, se combina el atractivo de realizar un playtest numerosas veces mediante IA automatizada y, además, esa IA evoluciona precisamente mediante los playtests y así cada vez puede dar feedback mejor del videojuego.

La simulación, en realidad, podría tener dos modos:

**Modo playest automatizado**

Se selecciona un preset de IA con algunas iteraciones que no recopilará datos para mejorar su comportamiento, sino que se centrará en su código actual y simplemente analizará información de la simulación.

**Modo aprendizaje**

Se aplica el proceso de deep learning para crear un preset de IA con más iteraciones y se ignora el tracking de resultados del juego.

A todo esto, el usuario podría concretar el número de iteraciones que a realizar por la IA en una misma ejecución del proceso.

**C# Machine Learning:**

<https://towardsdatascience.com/introduction-to-machine-learning-in-c-with-ml-net-bf45502d8110>

Este paquete se sumaría como dependencias, las cuales serían, hasta este momento:

* Controller Unity
* Unity Testing Framework (a elegir)
* Machine Learning

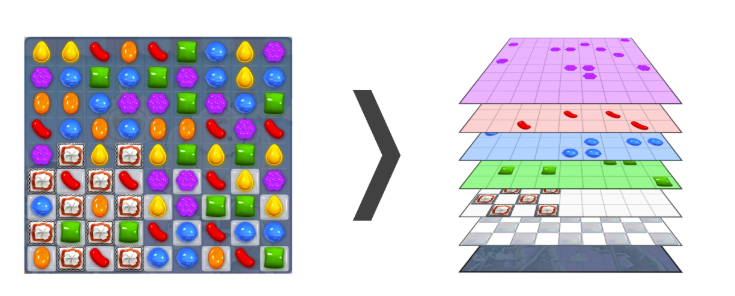
Y entonces… Cuál sería mi trabajo? Si me hago valer de un framework de machine learning, el trabajo aquí en realidad es montar todos los packages y encarar más el foco de trabajo a nivel de diseño y en cómo dar feedback de las simulaciones. Existiría parte de programación del vínculo de todas estas herramientas pero habría un gran diseño detrás para que se pueda aplicar en un entorno de un nivel de videojuegos.

A decir verdad, el script base de la IA debe ser programado para "dárselo como combustible" al deep learning (?)

**Posterior a la simulación**

Toda la información recolectada por el paquete sería simple información que se proporciona al usuario, a menos que el propio paquete cuente con la funcionalidad de realizar propuestas de mejora.

**Aún más datos**

****

<https://medium.com/techking/human-like-playtesting-with-deep-learning-92adafffe921>

King filtra la información del mapa en distintas grids, lo cual simplifica los cálculos y puede ser interesante para enfocar el proceso en un juego basado en un mapa con una grid, ya sea visual, imaginaria o que simplemente exista para estos cálculos.

En la aplicación se podría debugar y alternar las grids con todo tipo de información.

**Idea #3**

Sería combinar el apartado de la generación del nivel con el del testing automatizado. El punto fuerte sería el modo en el que desde un principio el nivel y la IA que lo recorre estarían hechos a medida.

Un paquete de Unity que sea capaz de organizar un nivel según los inputs y los assets que proporciones y que a su vez te permita seleccionar una IA para testearlo y que de feedback inteligente.

**Idea #4 (esto es divagación pura)**

Realidad aumentada, realidad virtual… ¿Cuál es el siguiente paso, el futuro, del grado de inmersión de un videojuego?

¿Y si el videojuego te juega a ti? Es decir, ¿y si TÚ eres el videojuego?

*Nota: Explorar esta idea a ver donde lleva*



En realidad, el concepto primitivo existe, se produce cada vez que la IA intenta adivinar tus acciones o pensamientos, lo cual sucede en una infinidad de juegos. Aunque tú eres el que se adentra en el contexto en el que la IA existe y te riges por sus normas, **y no a la inversa.**

Existe otro tipo de conexión con la vida real y es cuando tus actos en la vida real tienen consecuencias en el juego. Un juego que usa el concepto es Pokemon Go (y en realidad cualquier juego de RA o RV), donde te haces valer del contexto real para lograr objetivos en el mundo virtual, **y no a la inversa.**

¿Qué pasaría si yo disparo y asesino a un NPC en el juego y ~~muere en la vida real~~ tiene consecuencias en la vida real?

Es decir, en la actualidad no existen elementos en videojuegos que realmente impacten al usuario de un modo persistente en la vida real, dependiendo de sus acciones, menos:

* Perder dinero mediante el gambling y en general cualquier acción que involucre información del jugador
* Cualquier sensación producida por el juego que pueda llevar a la persona a un estado mental determinado (eg depresión, alegría)

Los jugadores además no realizan ningún “sacrificio” (aparte de los memes evidentes de *perder la vida social* etc) al empezar a jugar o en ciertos momentos del juego por tal de obtener X resultado (tampoco es como si te fueras a *arrancar un brazo en la vida real*…)

TODO: llevar esto hacia a algún lugar legal...

...En cuanto al tema del vínculo inmersivo entre jugador y juego, existe un campo interesante donde explotarlo, y es el de los juegos de terror. Se desarrollaría un aparato que pueda eg medir las pulsaciones del usuario en ciertas situaciones, o usar el eye-tracking para ver dónde mira y cuando aparta la mirada, por tal de recopilar esta información y mediante machine learning poder crear eventos dinámicos dentro del juego especialmente diseñados para hacer llorar al jugador como una niña de 3 años.

Esto tiene los vibes de Monstruos SA y el concepto de que a cada niño se le puede asignar un monstruo ideal que lo asuste para que chille lo máximo posible (esto se está volviendo poco-legal de nuevo…)

Y con esto se llegamos la idea #5

**Idea #5**

El concepto de modificar el nivel y los eventos que ocurren en él dependiendo de las acciones del jugador mediante machine learning.

Un ejemplo de este tipo de acción, de un modo scripted, sin que dependa del jugador, es el spawn de un enemigo off-camera:

<https://www.youtube.com/watch?v=nkaOItRQQR8>

Otro ejemplo más vinculado el mapa sería cómo se articula de un modo distinto a tu paso:

<https://www.youtube.com/watch?v=oCR1gAurnoI>

Estos ejemplos serían llevados a cabo gracias a un profiling que se haría del jugador. Por ende, el juego tendría asociado el atractivo de que cada partida/run es independiente del resto y lo más diferenciada posible

Este concepto se puede extrapolar y tener un sistema donde toda la experiencia (es decir un cúmulo de otros subsistemas) gire entorno a él. Por así decirlo, sería un nuevo módulo o manager.

La idea anterior de vincularlo con el terror puede funcionar.

¿Cómo centrarlo en level design?

**PROPUESTA FINAL CON OBJETIVOS:**

* **“Playtesting automatizado orientado al diseño de nivel”**

**Propuesta**

**Plataforma:**

Unity (se exporta como paquete)

**Dependencias:**

* Controller de Unity
* Unity Testing (u otro de framework de Testing)
* AI package (Learning Agents?)

**Workflow:**

* El usuario tiene su mapa montado en unity con objetos, los cuales debe proporcionar al inspector de uno de los scripts del paquete
* El personaje tiene dos scripts: el de la IA básica y la adaptación mía para reconocer el entorno del nivel
* El usuario puede definir dos tipos de variables:
* Del nivel: ubicación enemigos, walkabilidad, ubicación de ítems…
* Del jugador: tipología de Bartle, habilidad de jugador… Se tiene en cuenta la psicología de jugador, no es sólo un personaje

Cada tipo se introduce en uno de los 2 scripts propios del paquete

* El código del paquete, del script del personaje, previamente tendría “etiquetas” embedded de Unity Testing, es decir variables que comprobar durante la simulación
* El paquete simula la navegación paralelamente de múltiples personajes mediante el Unity Testing. Todos los personajes tendrían el objetivo principal de completar el nivel (se le define el punto de salida)
* Cada simulación aporta resultados en función del Unity Testing de las variables a comprobar (embedded). Éstas y el script del personaje que las contiene se verán afectados por el input del usuario de personaje + nivel
* Todos los datos recopilados se usarían para proporcionar una pantalla final detallada que otorgase calificaciones en cuanto a diseño de nivel de modo que se indique visualmente los errores y aciertos, con gráficos y con pop-ups explicativos de cada apartado de resultados
* Como extra, se realizan propuestas de mejoras a demanda del usuario, es decir, la aplicación re-interpreta los datos para generar cambios visibles en el nivel y mostrarlos en una nueva pantalla

**Objetivos**

* Crear un entorno de playtesting automatizado
* Proporcionar un paquete útil en un entorno accesible para el campo de videojuegos (Unity Store) que sea suficientemente genérico para adaptarse a propuestas de juego diversas
* Dar feedback útil y veraz a un desarrollador de nivel, de modo que obtenga información transparente en cuanto a la metodología empleada y los resultados
* Aplicar teorías de psicología de jugador y de diseño de niveles en el proceso

**Profesor**

Joan Pons, creo que es el director adecuado según:

* Conocimiento: Level Design
* Conocimiento: Psicología de jugador
* Conocimiento: Unity
* Experiencia en el sector
* Persona que muestra un interés y un entusiasmo genuinos en lo que hace