Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Aplicación de técnicas de inteligencia artificial en videojuegos: Generación de una librería de IA de Utilidad en Unity

Application of artificial intelligence techniques in video games: Generation of a Utility AI library in Unity.

José Javier Caraballo Cova

La Laguna, 11 de julio de 2024

D. Rafael Arnay del Arco, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Aplicación de técnicas de inteligencia artificial en videojuegos: Generación de una librería de IA de Utilidad en Unity”.

Ha sido realizada bajo su dirección por **D**. **José Javier Caraballo Cova**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de julio de 2024.

Agradecimientos

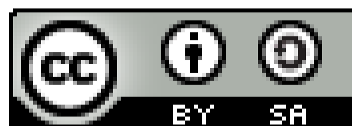
A mi familia y amigos por apoyarme a pesar de que muchas veces
me cuesta expresar su cariño de vuelta.

A mis profesores por sus enseñanzas durante la carrera a pesar
de que faltara a más de alguna clase.

A mi tutor por su ayuda y comprensión a lo largo de este proyecto
con el que (al fin) cierro mi etapa universitaria.

A todo el mundo al que le gustan los gráficos por computador.

Licencia



© Esta obra está bajo una licencia de Creative Commons
Reconocimiento-CompartirIgual 4.0 Internacional.

Resumen

El objetivo de este trabajo de fin de grado se centra en desarrollar un sistema de IA de utilidad. Con ello, se pretende proporcionar una herramienta eficiente y flexible para su uso en videojuegos o simulaciones. La IA de utilidad se basa en la evaluación y comparación de las posibles acciones de un agente o personaje no jugador (Non-Playable Character, NPC), asignando un valor de utilidad a cada acción posible y seleccionando aquellas con mayor valor.

Este tipo de metodología da realismo a las acciones de los NPCs permitiendo que tengan en cuenta los elementos que conforman su entorno y tomen decisiones más complejas y adaptativas, creando así experiencias de usuario con mayor inmersión y dinamismo.

Las tecnologías utilizadas para desarrollar este trabajo han sido el motor de videojuegos multiplataforma Unity y el editor de texto Visual Studio Code, Además se ha utilizado Git y Gitlab para tratar de llevar un control de versiones junto con una plataforma en la nube donde almacenarlo.

Palabras clave: Inteligencia artificial, Videojuegos, Unity, NPC, Utilidad.

Abstract

The objective of this final degree project is focused on developing a Utility AI system. With this, it is intended to provide an efficient and flexible tool for use in videogames or simulations. Utility AI is based on the evaluation and comparison of the possible actions of an agent or Non-Playable Character (NPC), assigning a utility value to each possible action and selecting those with the highest value.

This type of methodology gives realism to the actions of the NPCs allowing them to take into account the elements that make up their environment and make more complex and adaptive decisions, thus creating user experiences with greater immersion and dynamism.

The technologies used to develop this project have been the multiplatform video game engine Unity and the text editor Visual Studio Code. Git and Gitlab have also been used to try to keep a version control along with a cloud platform where to store it.

Keywords: Videogames, Unity, NPC,, Utility, AI.

Índice

Capítulo 1 Introducción.......... 9
1.1 Antecedentes y estado actual.......... 10
Capítulo 2 Diseño y desarrollo.......... 11
2.1 Tecnologías utilizadas.......... 11
2.1.1 Unity.......... 12
2.1.2 Visual Studio Code.......... 12
2.1.3 Git.......... 12
2.1.4 Gitlab.......... 12
2.2 Diseño.......... 13
2.3 Estructura.......... 14
2.3.1 Core.......... 15
2.3.2 Items.......... 19
2.3.3 Actions.......... 20
2.3.4 UI.......... 21
2.3.5 Utils.......... 24
Capítulo 3 Caso práctico.......... 24
3.1 Configuración del entorno.......... 24
3.2 Ejecución y resultados.......... 28
Capítulo 4 Conclusiones y futuras líneas de trabajo.......... 30
4.1 Futuras líneas de trabajo.......... 30
Chapter 5 Summary and future lines of work.......... 31
5.1 Future lines of work.......... 31
Capítulo 6 Presupuesto.......... 32
Capítulo 7 Bibliografía.......... 33

Índice de figuras

Capítulo 1 Introducción ... 9
Figura 1: Ciclo de un sistema de IA de utilidad ... 10
Capítulo 2 Diseño y desarrollo ... 11
Figura 2.1: Estructura interna del proyecto de Unity ... 14
Figura 2.2: Diagrama de clases del módulo Core ... 15
Figura 2.3: Función GetAdvertisableItems() de Context ... 16
Figura 2.4: Ejemplo de la curva de la necesidad Hunger ... 17
Figura 2.5: Funciones EvaluateItems() y ScoreItem() de Brain ... 18
Figura 2.6: Diagrama de clases del módulo Items ... 19
Figura 2.7: Pseudocódigo de la función Use() ... 20
Figura 2.8: Diagrama de clases del módulo Action ... 21
Figura 2.9: Diagrama de clases del módulo UI ... 22
Figura 2.10: Interfaz DataWindow ... 22
Figura 2.11: Burbuja de texto sobre un ítem ... 23
Figura 2.12: Burbuja de texto sobre un NPC ... 23
Figura 2.13: Diagrama de clases del módulo Utils ... 24
Capítulo 3 Caso práctico ... 24
Figura 3.1: Escena de Unity ... 25
Figura 3.2: Comparación entre una curva lineal y exponencial ... 26
Figura 3.3: Curvas de las necesidades ... 26
Figura 3.4: Distribución de necesidades para cada NPC ... 28
Figura 3.5 Distribución de ítems para cada NPC ... 29
Figura 3.6: Distribución de acciones para cada NPC ... 29

Índice de tablas

Capítulo 3 Caso práctico **24**
Tabla 1: Ítems de la escena 27
Capítulo 6 Presupuesto **32**
Tabla 2: Presupuesto 32-33

Capítulo 1 Introducción

En el mundo de los videojuegos, es cada vez más creciente la demanda de experiencias más inmersivas y envolventes. Esto ha dado lugar a que durante los últimos años, los desarrolladores de videojuegos se enfrenten al desafío de crear mundos no solo atractivos visualmente sino que también ofrezcan una mejor experiencia adaptable e interactiva entre jugadores, NPCs [1] y el mundo que los rodea.

En este contexto, el campo de la inteligencia artificial (IA)[2] se erige como una herramienta fundamental para abordar los problemas que acarrea esta demanda. Son diferentes las técnicas utilizadas a lo largo de los años para solventarlos. Comenzando con las máquinas de estados (Finite State Machines) [3] y Behaviour Trees [4], hasta llegar a técnicas más avanzadas como el BDI [5] (Beliefs, Desires, Intentions) o GOAP [6] (Goal-Oriented Action Planning) donde cada enfoque ha tratado de superar las limitaciones / carencias de su predecesor.

Para suplir la mencionada demanda, la IA de utilidad se presenta como una técnica efectiva y versátil. A diferencia de las anteriores mencionadas, que pueden dar lugar a NPCs “rígidos” y predecibles, la IA de utilidad permite a los personajes tomar decisiones adaptativas y variadas en función del contexto del juego. Esta técnica se basa en la asignación de valores de “utilidad” a las diferentes acciones que un NPC puede realizar, evaluando el estado general del juego y del NPC en sí. Esto da lugar a una mejor toma de decisiones y por lo tanto la sensación de tener NPCs más inteligentes, reactivos y “conscientes” de su alrededor. Otro aspecto positivo de este sistema es que es fácilmente combinable con las técnicas mencionadas anteriormente, permitiendo fortalecer sus puntos débiles y creando un sistema de inteligencia artificial más robusto. Mejorando el realismo y por lo tanto la experiencia de usuario.

A alto nivel, podríamos definir que que un sistema de IA de utilidad se resume en los siguientes pasos:

- **Evaluación del contexto**: El NPC evalúa su estado actual y el contexto de la escena, calculando la “utilidad” de cada acción posible.

- **Generación de una lista de puntuaciones**: Una vez realizado el cálculo de puntuación de todas las acciones, se genera una lista ordenada de mayor a menor puntuación.

- **Selección de la acción**: Se selecciona la acción con mayor puntuación de la lista previa o una de las que tenga mayor puntuación.

- **Ejecución de la acción**: El NPC ejecuta la acción seleccionada, lo que conlleva cambios tanto en su estado como en el mundo que lo rodea. Tras ello, el

ciclo vuelve a comenzar, permitiendo una adaptación continua y dinámica del comportamiento del NPC.

Este proceso se ilustra en la siguiente Figura 1:

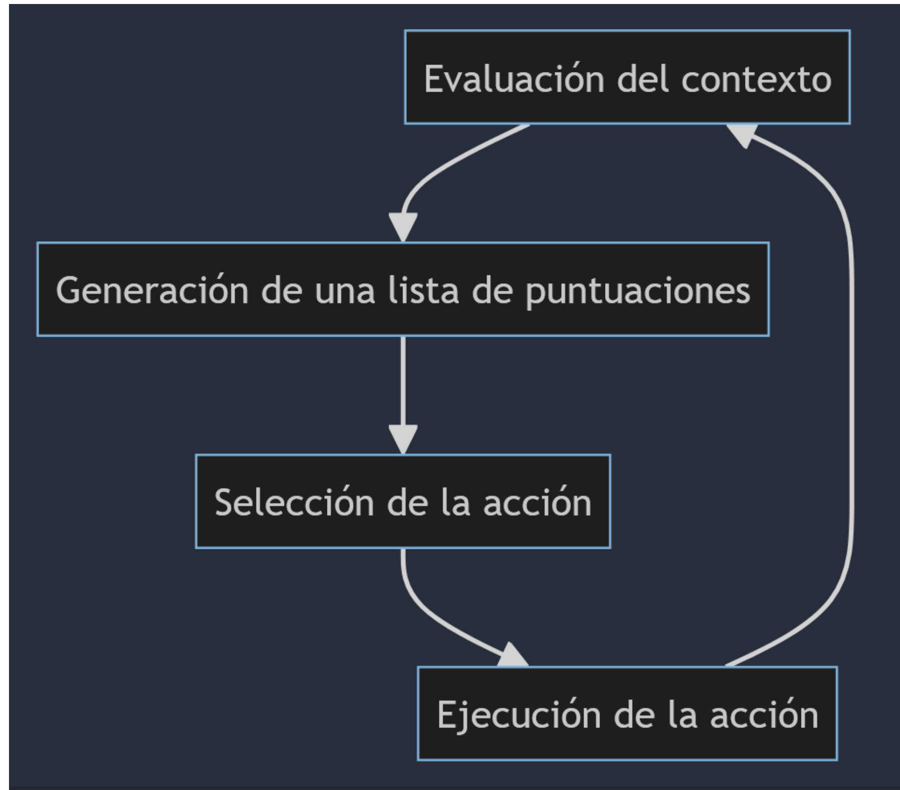


Figura 1: Ciclo de un sistema de IA de utilidad.

El objetivo de este trabajo es implementar una librería de IA de utilidad en Unity, sencilla, modular y ampliable. Además de generar y evaluar su funcionamiento a través de un caso de uso para demostrar las distintas funcionalidades de la herramienta.

1.1 Antecedentes y estado actual

Videojuegos como la saga de The Sims [7] o Dragons Age: Inquisition [8] son ejemplos en donde se ha implementado y demostrado el uso y capacidad de la IA de utilidad. En el caso de The Sims, la implementación de la IA de utilidad es evidente en la manera en que los personajes toman decisiones en función de sus necesidades. Cada Sim evalúa constantemente múltiples necesidades, como hambre, sueño, interacciones sociales, trabajo, etc... asignando prioridades basadas en la utilidad percibida.

Esta adaptabilidad en la toma de decisiones refleja que las acciones de los Sims están impulsadas por la maximización de su bienestar general. En la entrega de Dragons Age, la IA de utilidad se manifiesta en la forma en que los NPC's

compañeros interactúan con los enemigos durante una pelea. Cada NPC tiene características como vida, estamina o maná y durante la batalla, debe decidir qué acción debe realizar teniendo en cuenta el estado de estas y la situación del grupo en el campo de batalla.

Otro videojuego donde podemos encontrar IA de utilidad es Total War: Three Kingdoms [9]. Este videojuego utiliza un sistema de “diplomacia” para crear alianzas entre las distintas facciones del juego. Dicho sistema está pensado para tener en cuenta las características de las facciones involucradas junto con el estado actual del mapa y así crear una serie de tratados. Estos tratados conllevan a una serie de acciones y son estas las que son evaluadas mediante IA de utilidad. Esta implementación permite a las facciones tomar decisiones estratégicas que no solo benefician sus intereses inmediatos, sino que también tienen en cuenta las repercusiones a largo plazo. Por ejemplo, una facción podría decidir aliarse temporalmente con un enemigo para enfrentarse a una amenaza mayor, evaluando así la utilidad de la cooperación.

Una búsqueda rápida en internet nos lleva a encontrar que en la Unity Asset Store podemos encontrar distintas implementaciones de esta técnica, como por ejemplo Apex Utility AI [10] o CrystalAI [11]. A su vez, podemos encontrar implementaciones en otros motores de videojuegos como Utility AI for Godot [12] para Godot [13] o Easy Utility AI [14] para Unreal Engine [15].

Capítulo 2 Diseño y desarrollo

Este capítulo se centrará en explicar todo lo relacionado con el diseño, estructura y programación de la librería. El desarrollo de la librería de IA de utilidad comenzó con una fase de búsqueda de información. A través de distintas fuentes como libros [16], posts en internet [17] y videos [18] fui capaz de hacerme una idea sobre cuál es el concepto principal sobre el que se centra un sistema de IA de utilidad.

Posteriormente, se pasó al diseño del sistema, el cual se basó en un marco modular que permitiera una fácil integración y expansión de todos los elementos necesarios para que el sistema funcione. Inspirándonos en el enfoque utilizado en los juegos de la serie *The Sims*, se crearon módulos específicos para gestionar las diferentes necesidades de los NPCs, las acciones, la evaluación y uso de ítems, etc… Además, se han implementado algunas clases para mostrar información importante durante la ejecución y otras utilidades.

La implementación de la librería se desarrolló con el lenguaje de programación C# [19] y el motor de videojuegos Unity [20].

2.1 Tecnologías utilizadas

Las tecnologías utilizadas para llevar a cabo todo este proyecto fueron:

2.1.1 Unity

Unity es un motor de videojuegos y aplicaciones en tiempo real creado por Unity Technologies. Su versatilidad permite a desarrolladores crear increíbles aventuras y simulaciones tanto 2D como 3D. Debido a esto, se ha convertido en una de las herramientas más populares tanto entre estudios de videojuegos como desarrolladores indie. Además del sector de videojuegos, Unity es también utilizado en otros sectores como el cine o la arquitectura.

2.1.2 Visual Studio Code

Visual Studio Code [21] es un editor de texto multiplataforma desarrollado por Microsoft. Viene con soporte integrado para JavaScript, TypeScript o Node.js y tiene un rico ecosistema de extensiones para otros lenguajes y tiempos de ejecución.

Escogí usar Visual Studio Code frente a un IDE como Visual Studio [22] (el cual puede ser instalado junto con Unity) debido a mi familiaridad con dicho editor ya que es la herramienta con la que escribo código desde hace unos años.

2.1.3 Git

Git [23] es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para gestionar con rapidez y eficacia desde proyectos pequeños hasta muy grandes. Fue creado por Linus Torvalds y lanzado en el año 2005. Git permite seguir un histórico de los cambios en el código fuente, facilitando la colaboración entre múltiples desarrolladores. Al ser distribuido, cada desarrollador tiene una copia completa del historial del proyecto, lo que permite trabajar de manera independiente y sin conexión a un servidor central.

2.1.4 Gitlab

GitLab [24] es una plataforma de DevOps que proporciona una suite integrada de herramientas para gestionar todo el ciclo de vida del desarrollo de software, desde la planificación y el desarrollo hasta la integración continua (CI), la entrega continua (CD) y el monitoreo. Fue fundada en 2011 por los programadores ucranianos Dmitriy Zaporozhets y Valery Sizov. GitLab ha evolucionado significativamente, ofreciendo tanto versiones gratuitas como de pago que se adaptan a las necesidades de equipos de todos los tamaños.

GitLab proporciona un control de versiones robusto basado en Git, permitiendo a los equipos colaborar de manera efectiva. Las ramas y las fusiones se gestionan con facilidad, y la revisión de código se simplifica a través de merge requests, donde

los desarrolladores pueden revisar, discutir y aprobar cambios antes de integrarlos en la rama principal.

2.2 Diseño

Como mencionamos anteriormente, el diseño de esta librería bebe de la implementación de la saga de videojuegos The Sims. En este sistema, los NPCs tienen diversas necesidades o motivos, como hambre, energía, diversión o higiene, que van desde 100 hasta 0. Las necesidades se van agotando constantemente y es responsabilidad del NPC evaluar su entorno y tomar decisiones para satisfacerlas.

Lo que caracteriza el sistema de The Sims frente a otros sistemas de IA de utilidad, es en cómo funciona el proceso de elección y evaluación para realizar una acción. En The Sims (y por lo tanto en nuestra librería), el proceso de elección se realiza a través de los objetos (ítems) de la escena. Son los objetos quienes “anuncian” qué necesidad pueden satisfacer y, cuando son escogidos, es cuando se realiza la acción asociada a ellos. Por ejemplo, una cama puede ofrecer mejorar en diez puntos la energía, mientras que una manzana puede mejorar en veinte el hambre.

Cuando un NPC necesita tomar una decisión, primero hace una lista de todos los objetos en la escena y lo que cada uno ofrece. Luego, el NPC pondera estos anuncios basándose en sus necesidades actuales. A continuación, el NPC ordena las interacciones disponibles según las puntuaciones ponderadas y elige la que proporcionará el mayor beneficio en ese momento. Para evitar comportamientos robóticos, el NPC no siempre elige la mejor opción, sino que selecciona uno de los ítems mejor puntuados al azar, lo que introduce una variabilidad en su comportamiento.

Además, no todas las necesidades tienen la misma importancia. Por ejemplo, si un NPC está hambriento y aburrido, es más probable que elija comer antes que entretenerse. Esto se logra definiendo curvas específicas para cada necesidad. Las necesidades fisiológicas como el hambre o la vejiga tienen curvas que priorizan su satisfacción en niveles bajos, mientras que necesidades menos cruciales como la diversión o el confort tiene curvas con menor pendiente, indicando que estas necesidades se vuelven más importantes solo cuando las necesidades fisiológicas están relativamente satisfechas.

Este funcionamiento está inspirado en la pirámide de necesidades de Maslow [25], la cual asegura que los NPCs se concentren en tareas cognitivas y sociales sólo cuando sus necesidades más básicas están satisfechas. En el videojuego

original, los Sims también tienen personalidades las cuales afectan a esta toma de decisiones, pero esto no lo tendremos en cuenta en nuestra implementación.

2.3 Estructura

Internamente, hemos organizado el proyecto de Unity tal y como se puede ver en la Figura 2.1:

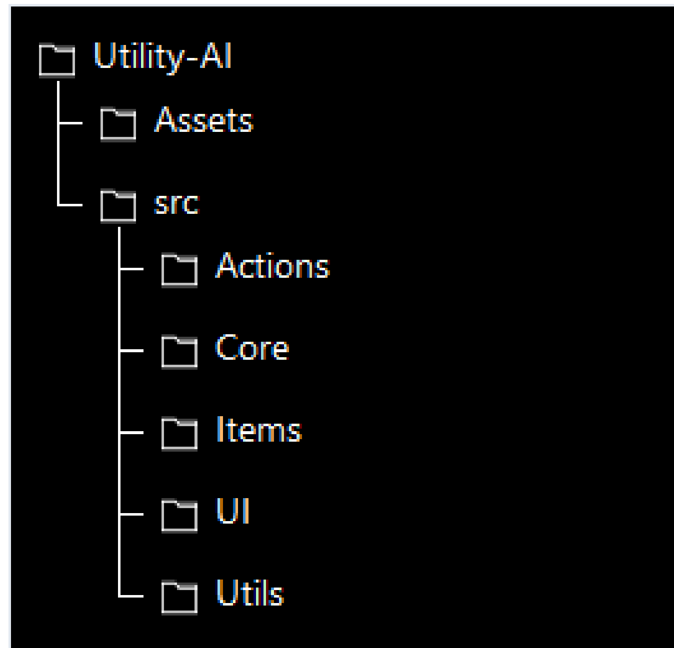


Figura 2.1: Estructura interna del proyecto de Unity.

- **Assets**: directorio principal que contiene todos los recursos del proyecto, como los modelos, animaciones, texturas, etc…
- **src**: directorio que contiene todo el código fuente del proyecto, subdividido en varios según su funcionalidad.
    - **Actions**: en este directorio se encuentran las distintas acciones que pueden realizar los NPCs.
    - **Core**: contiene las clases fundamentales del sistema, como *NpcController*, *Brain* o *Context*. Estas clases gestionan la lógica principal del comportamiento de los NPCs.
    - **Items**: aquí podemos encontrar las clases que definen los ítems del juego.
    - **UI**: el directorio UI contiene todo lo relacionado con mostrar información en el juego .

○ **Utils**: contiene utilidades auxiliares para otros módulos.

En las siguientes secciones se explicará en detalle cada uno de estos módulos.

2.3.1 Core

El núcleo de nuestro sistema está compuesto por varias clases fundamentales que gestionan la lógica principal del comportamiento de los NPCs. Estas clases son responsables de la toma de decisiones, la evaluación del entorno y la ejecución de acciones, asegurando que los NPCs actúen de manera coherente y adaptativa en respuesta a sus necesidades.

En la Figura 2.2 podemos observar el diagrama de clases de este módulo.

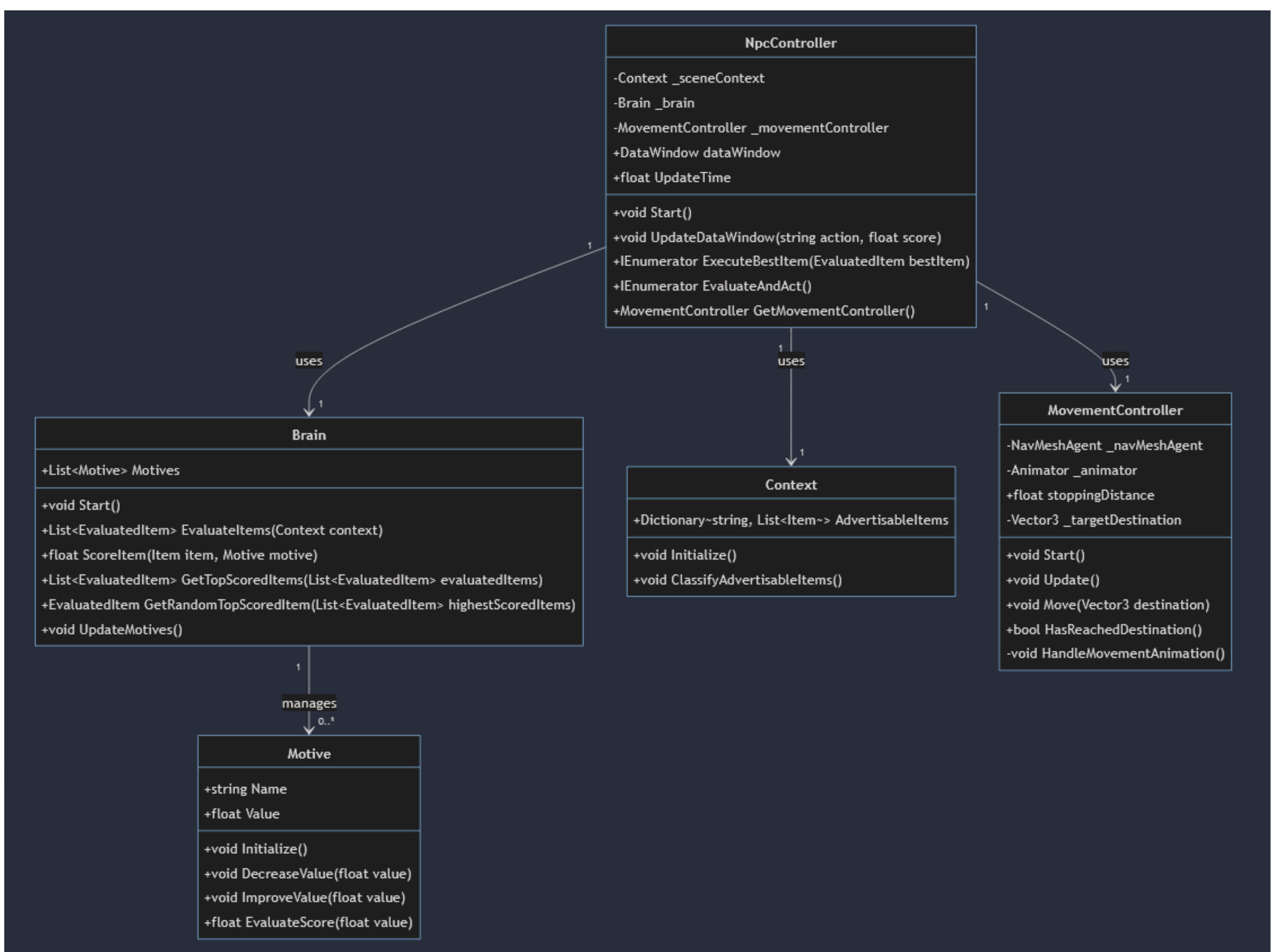


Figura 2.2: Diagrama de clases del módulo Core.

La clase ***Context*** organiza y gestiona los ítems del entorno del juego según la necesidad que satisfacen. Proporciona a los NPCs la información necesaria para evaluar su entorno y tomar decisiones basadas en la utilidad de los ítems disponibles. Los ítems son evaluados constantemente durante el bucle del juego,

creando así un adaptación continua y dinámica a los posibles cambios. En la Figura 2.3 podemos ver cómo se implementa el algoritmo para la gestión y evaluación continua de los ítems en nuestra librería.

Figura 2.3: Función *GetAdvertisableItems()* de Context.

Las **necesidades o *motives*** representan las necesidades básicas y deseos de los NPCs en el juego. Estas son fundamentales para guiar el comportamiento de los NPCs, permitiéndoles tomar decisiones basadas en sus valores actuales. Durante el bucle del juego, las necesidades disminuyen con el tiempo y tras utilizar un ítem es cuando su valor aumenta.

Un aspecto clave de las necesidades, es que estas poseen una curva (*Motive Curve*), la cual nos sirve para representar la urgencia de estas. La urgencia se evalúa a través de la curva, que toma el valor actual de la necesidad y lo transforma en una puntuación de urgencia. Estas curvas pueden retocarse individualmente para ajustar la urgencia de cada necesidad, permitiendo a los diseñadores obtener distintos comportamientos en los NPCs según sea necesario. En la Figura 2.4 podemos ver un ejemplo de la curva de la necesidad *"Hunger"*. Cuando un NPC no tiene mucha hambre (el valor normalizado en el eje de las abscisas es muy cercano a uno), la puntuación de urgencia (el valor en el eje de las ordenadas) es casi cero. Sin embargo, a medida que el valor de hambre disminuye (X se acerca a 0), la urgencia (Y) aumenta exponencialmente.

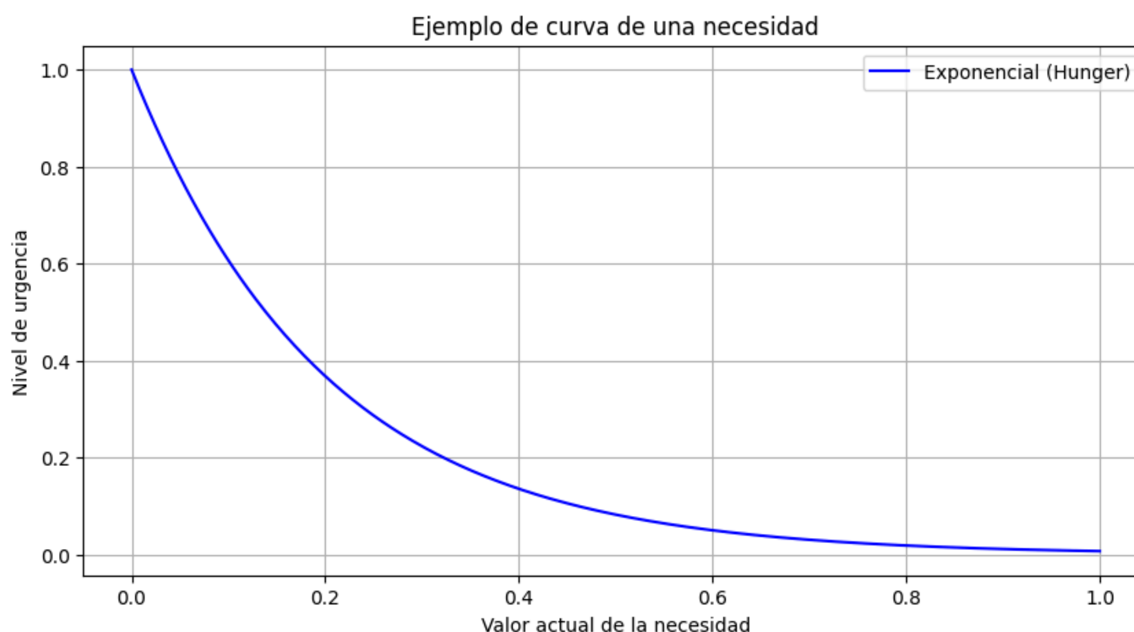


Figura 2.4: Ejemplo de la curva de la necesidad Hunger.

Las necesidades son implementadas haciendo uso de los ScriptableObjects [26] de Unity, los cuales permiten almacenar datos de forma eficiente y accesible. Esta característica facilita la creación y gestión de múltiples instancias de necesidades, asegurando que todas compartan una estructura base común. El uso de ScriptableObjects será algo que utilizaremos también al implementar otros elementos como los ítems o las acciones.

La clase ***Brain*** actúa como el núcleo de la toma de decisiones del NPC. Se encarga de evaluar las necesidades y los ítems disponibles en el entorno (contexto) para determinar las acciones más beneficiosas.

Con respecto al cálculo de utilidad de los ítems, se ha utilizado la siguiente fórmula:

$$Score = MotiveCurve(CurrentMotiveValue)\ x\ ItemValue$$

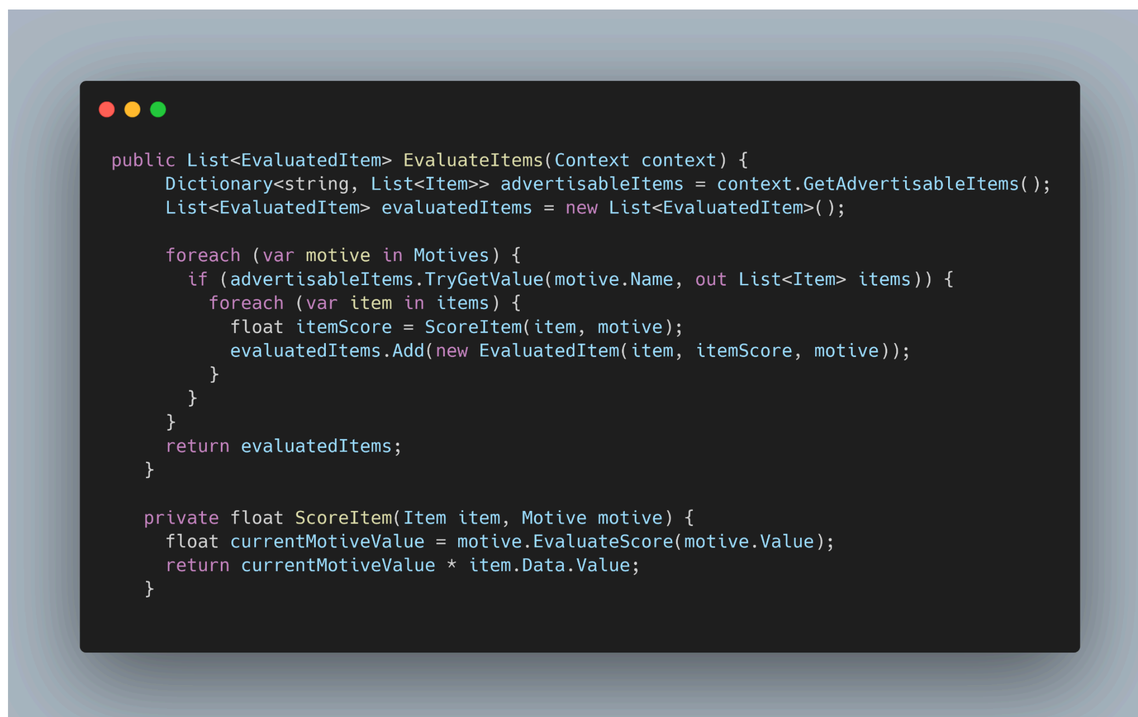
Donde:

- $MotiveCurve(CurrentMotiveValue)$: es la evaluación del valor actual de la necesidad en su curva.

- $ItemValue$: es el valor actual que el ítem proporciona para mejorar la necesidad que satisface.

Esta fórmula permite cuantificar la utilidad de cada ítem basado en la situación actual del NPC, considerando tanto la necesidad que satisface como el valor que aporta.

El proceso de evaluación se aglutina en la función *EvaluateItems*, que podemos ver en la Figura 2.5, la cual crea una lista con todos los ítems de la escena evaluados según sus puntuaciones de utilidad. Brain también se encarga de generar una lista ordenada de ítems con las tres puntuaciones más altas que superan cierto umbral. De estos tres ítems, se selecciona uno aleatoriamente para asegurar variedad en el comportamiento del NPC. Otra tarea de la que se encarga Brain, es de actualizar (decrementar) las necesidades de los NPCs durante el bucle del juego.



```
public List<EvaluatedItem> EvaluateItems(Context context) {
     Dictionary<string, List<Item>> advertisableItems = context.GetAdvertisableItems();
     List<EvaluatedItem> evaluatedItems = new List<EvaluatedItem>();

     foreach (var motive in Motives) {
       if (advertisableItems.TryGetValue(motive.Name, out List<Item> items)) {
         foreach (var item in items) {
           float itemScore = ScoreItem(item, motive);
           evaluatedItems.Add(new EvaluatedItem(item, itemScore, motive));
         }
       }
     }
     return evaluatedItems;
   }

   private float ScoreItem(Item item, Motive motive) {
     float currentMotiveValue = motive.EvaluateScore(motive.Value);
     return currentMotiveValue * item.Data.Value;
   }
```

Figura 2.5: Funciones EvaluateItems() *y* ScoreItem*()* de Brain*.*

La clase ***NpcController*** es la encargada de coordinar y gestionar las acciones de los NPCs dentro del juego. Esta clase interactúa con el contexto, el cerebro, y el controlador de movimiento para asegurar que los NPCs evalúan su entorno, seleccionan los ítems más adecuados y realizan las acciones correspondientes.

NpcController inicia el proceso de evaluación y acción mediante una corrutina que se ejecutará durante todo el bucle del juego. Esta corrutina utiliza los métodos de *Brain* para evaluar los ítems disponibles, seleccionar el mejor y ejecutar las acciones correspondientes. Además, la clase se encarga de actualizar la interfaz de usuario a través de la clase *DataWindow* y de generar burbujas de texto, usando la clase *ItemBubble*, que muestran información sobre cada ítem evaluado. Ambas clases serán explicadas en detalle más adelante.

Para manejar el movimiento de los NPCs, se ha creado la clase ***MovementController***, la cual se encarga de manejar la navegación de los personajes en el entorno del juego. Cuando un ítem es seleccionado, el NPC comprueba si está lo suficientemente cerca de él y si no lo está, establece una ruta

hacia él iniciando el movimiento. Esta clase hace uso de Unity NavMesh [27] una herramienta que permite a los NPCs desplazarse a través de un terreno pre-analizado y el cual elimina por completo la necesidad de programar un sistema de movimiento desde cero.

Esta clase no solo gestiona la dirección y el destino del movimiento, sino que también se asegura de que las animaciones correspondientes se activen en el momento adecuado. Por ejemplo, cuando un NPC se mueve hacia un ítem, se activa la animación de caminar, y al llegar a su destino, se detiene y cambia su animación al estado de “Idle”.

2.3.2 Items

Los ítems juegan un papel crucial, ya que representan las diferentes oportunidades y recursos disponibles para que los NPCs satisfagan sus necesidades. En nuestro sistema, hemos definido un Ítem como el conjunto de sus características y la acción asociada a él. Para gestionar estos ítems, hemos implementado las clases *ItemData*, *Item* y *EvaluatedItem*. La Figura 2.6 muestra el diagrama de clases de este módulo.

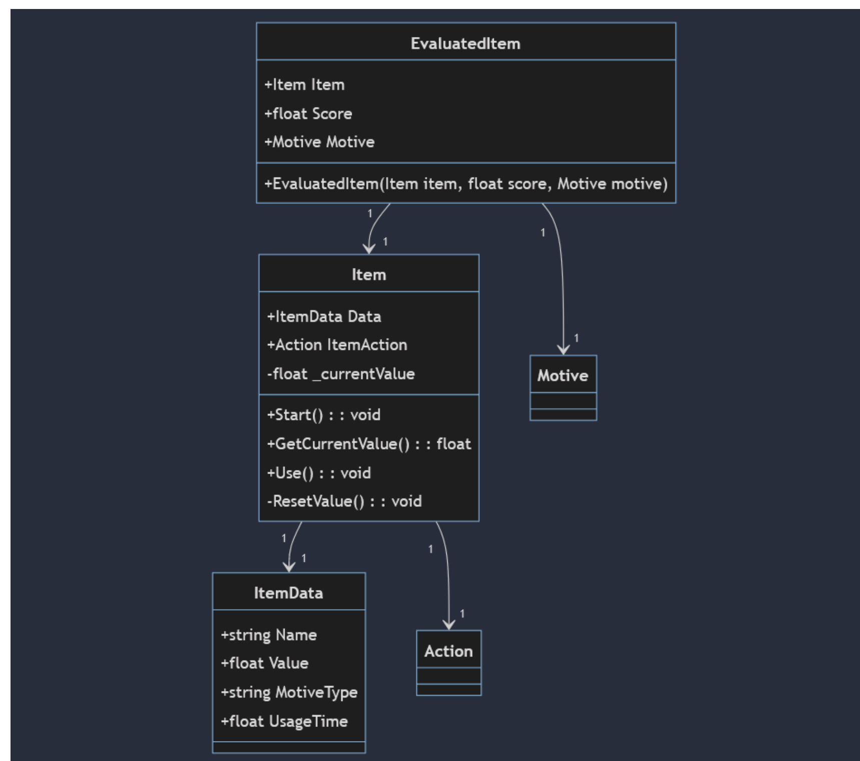


Figura 2.6: Diagrama de clases del módulo Items.

La clase **ItemData**, implementada como un ScriptableObject, define las características base de los ítems. *ItemData* proporciona una estructura consistente para todos los ítems, asegurando que todos compartan un conjunto común de propiedades, pero permitiendo que sus valores específicos varíen. Para nuestra

librería, hemos definido que las características de ítem son las siguientes:

- **Name**: el nombre del ítem.
- **Value**: es el valor inicial del ítem.
- **Motive Type**: es el motivo que satisface dicho ítem.
- **Usage Time**: es el tiempo que se tarda en usar un item. Será utilizado al realizar la acción asociada.

La clase ***Item*** actúa como el núcleo alrededor del cual giran las evaluaciones de utilidad y las decisiones de los NPCs, integrando las características definidas en *ItemData* junto con las acciones. Esta clase añade funcionalidades para la manipulación de los ítems durante el bucle del juego. Cada vez que un ítem es utilizado, su valor disminuye una cantidad aleatoria. Esto conlleva a que en la siguiente iteración del bucle de evaluación, el cálculo de utilidad de un ítem difiera del anterior calculado, permitiendo así que otros ítems que satisfacen la misma necesidad se vuelvan más atractivos. Cuando el valor de un ítem llega a cero, este se "resetea" y vuelve a tener su valor inicial. En la Figura 2.7 podemos ver el pseudocódigo de la función *Use* la cual implementa el funcionamiento descrito anteriormente.

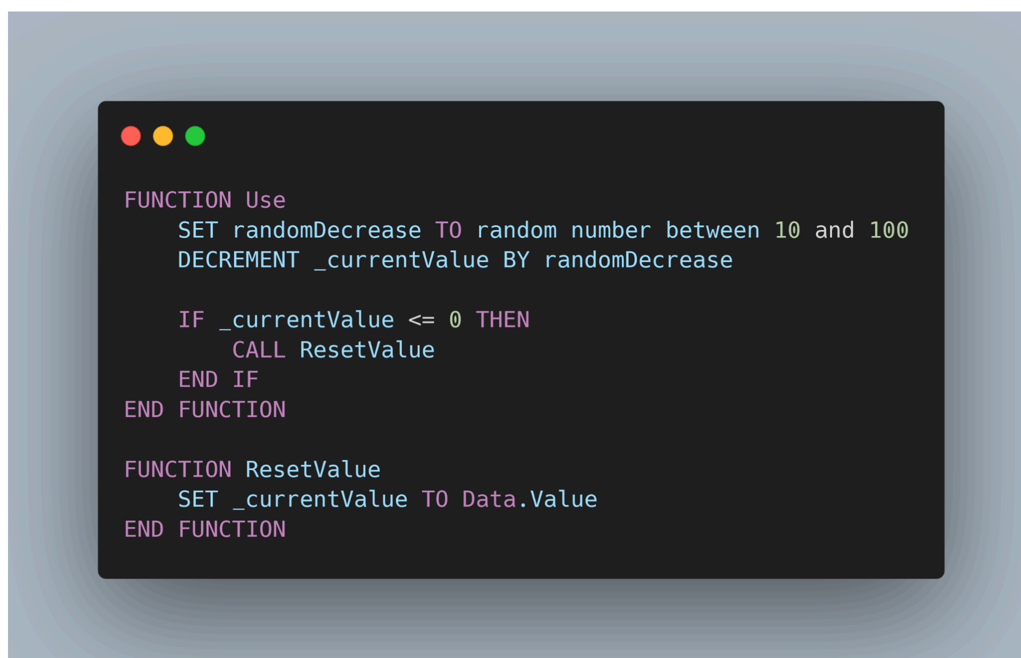


Figura 2.7: Pseudocódigo de la función Use().

La clase auxiliar ***EvaluatedItem*** se utiliza durante el cálculo de utilidad de los ítems. Almacena el ítem, su valor de utilidad y la necesidad que satisface. Esta estructura permite una fácil gestión y selección de los ítems más útiles para los NPCs en cada ciclo de evaluación.

2.3.3 Actions

Las acciones en nuestro sistema representan comportamientos específicos que

los NPCs pueden ejecutar en respuesta a sus necesidades. Estas acciones se implementan a través de la clase ***Action***.

Cuando un ítem es seleccionado, se lleva a cabo la acción asociada a él. Para que el NPC no realice la acción en el mismo sitio donde está, y para añadir realismo, antes comprobamos la ubicación del NPC frente al ítem escogido y nos acercamos hasta el si es necesario.

Además, cada acción no solo afecta al ítem en cuestión (disminuyendo su valor y por lo tanto su utilidad en la siguiente evaluación), sino que también tiene repercusiones en el estado de la necesidad correspondiente del NPC. Una vez realizada la acción, se actualiza el valor de la necesidad sumando el valor que ofrece dicho ítem, reflejando los cambios provocados por dicha interacción. Esto crea una dinámica donde el comportamiento del NPC es una respuesta directa a sus necesidades internas y a las oportunidades disponibles en su entorno, logrando así un comportamiento adaptativo y variado.

Las acciones también están diseñadas para proporcionar feedback visual al jugador. Por ejemplo, al ejecutar una acción, se genera una burbuja de texto sobre el NPC que indica el ítem utilizado, mejorando la comprensión del comportamiento del NPC por parte del jugador / desarrollador. La Figura 2.8 a continuación nos muestra la relación entre la clase ***Action*** y el resto del sistema.

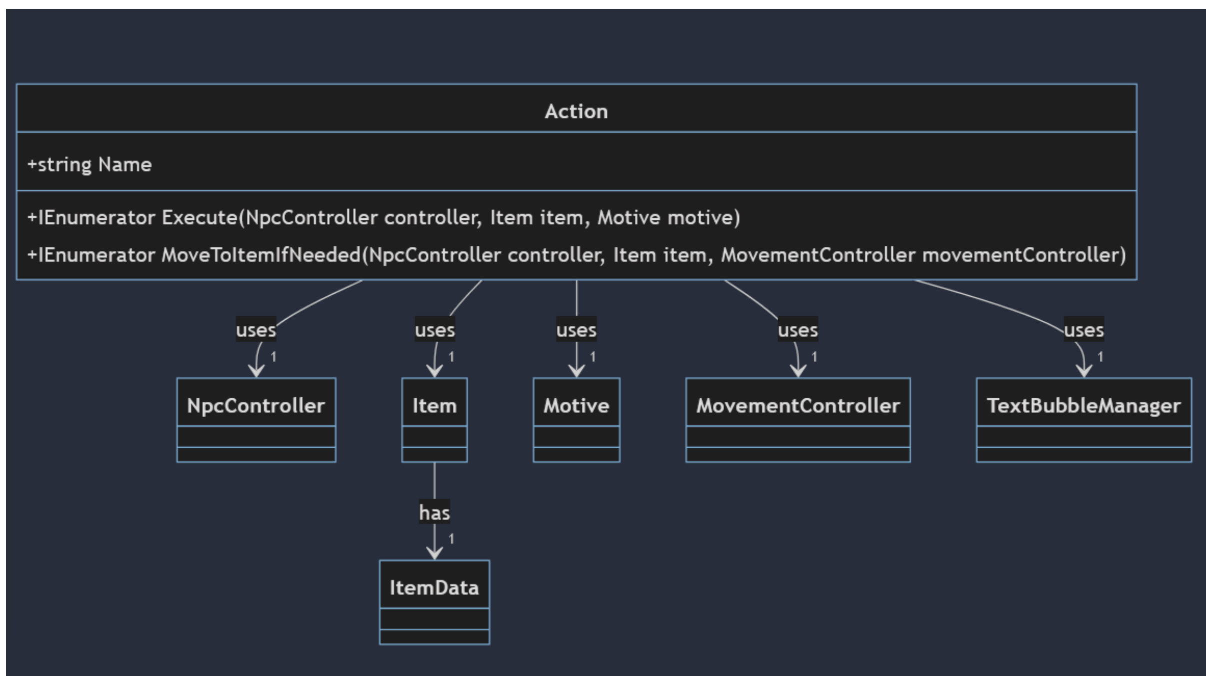


Figura 2.8: Diagrama de clases del módulo Action.

2.3.4 UI

El módulo UI contiene todas las clases relacionadas con la interfaz de usuario del sistema. Estas clases se encargan de mostrar información importante tanto para el jugador como para el desarrollador. La Figura 2.9, representa el diagrama de clases de este módulo.

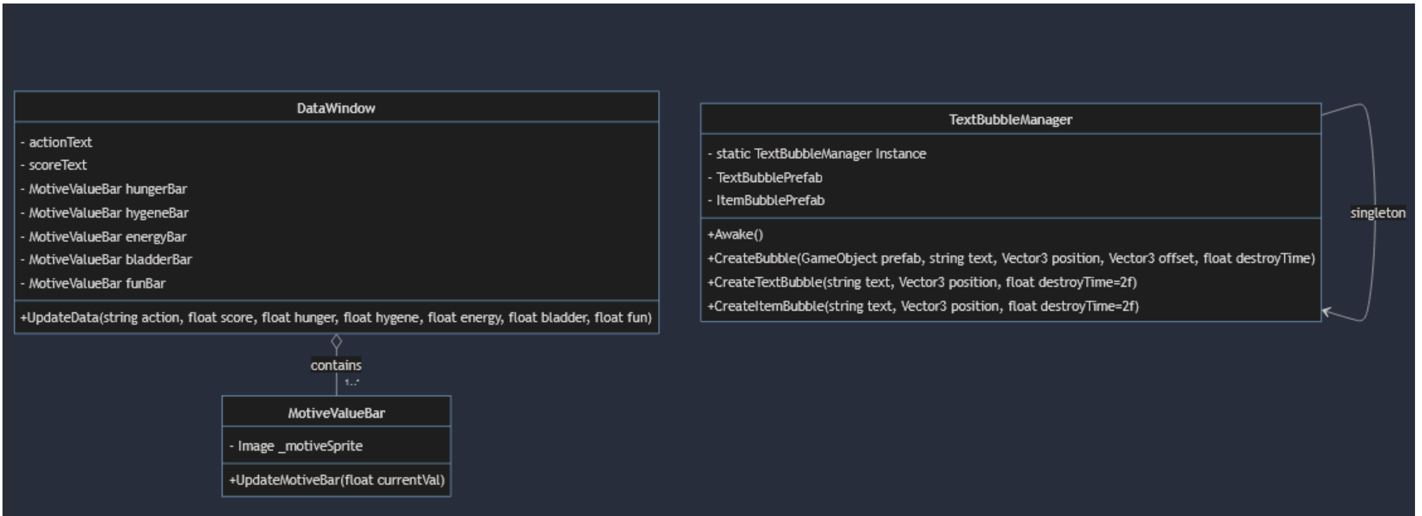


Figura 2.9: Diagrama de clases del módulo UI.

La clase ***DataWindow*** gestiona la ventana que muestra los valores actuales de las necesidades de los NPCs, junto con la acción escogida y la pertinente puntuación de utilidad. Hace uso de *MotiveValueBar* para mostrar los valores de las necesidades como "barras" las cuales se vacían o rellenan dependiendo de su estado. Se ha creado un prefab para que esta ventana sea reutilizable.

La clase ***MotiveValueBar*** gestiona la visualización gráfica de los valores de los motivos en forma de barras. La clase actualiza el relleno de una imagen según el valor actual del motivo, proporcionando una representación visual clara y fácil de interpretar del estado de los motivos del NPC. Al igual que *DataWindow*, se ha creado un prefab de esta "barra" para que sea reutilizable. A continuación, en la Figura 2.10 podemos ver cómo luce la interfaz final.

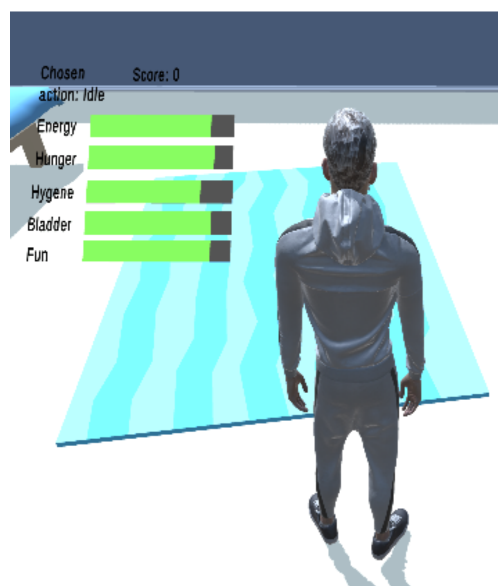


Figura 2.10 Interfaz DataWindow.

La clase ***TextBubbleManager*** administra la generación de burbujas de texto que aparecen durante unos segundos sobre los NPCs o sobre los ítems para luego desaparecer. En el caso de los ítems, estas burbujas muestran mensajes con el valor actual de este junto con las necesidad que cubren cada vez que son evaluados, Figura 2.11. Mientras que en el caso de la que se muestra sobre los NPCs, esta aparece cada vez que se utiliza un ítem (se realiza su acción asociada), Figura 2.12. Para cada burbuja se ha creado un prefab diferente, principalmente debido a diferencias en el tamaño que se quiere mostrar esta información.

Figura 2.11: Burbuja de texto sobre un ítem.

Figura 2.12: Burbuja de texto sobre un NPC.

2.3.5 Utils

En ***Utils*** se agrupan clases adicionales que, aunque no son cruciales para la librería, ofrecen funcionalidades que facilitan el desarrollo y la interacción con el juego. Estas clases no se explicarán en detalle, ya que están diseñadas principalmente para asistir al desarrollador y su uso es opcional para quienes deseen utilizar la librería. La Figura 2.13 ilustra el diagrama de clases de este módulo.

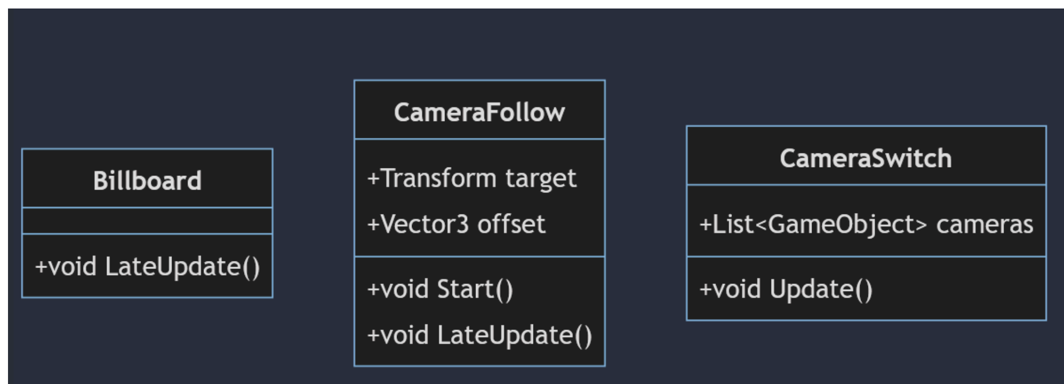


Figura 2.13: Diagrama de clases del módulo Utils*.*

CameraFollow gestiona el seguimiento de la cámara al NPC, manteniendo una vista constante, ***CameraSwitch*** permite cambiar entre diferentes cámaras en la escena con teclas específicas y, finalmente, ***Billboard*** asegura que ciertos elementos siempre miren hacia la cámara, mejorando la legibilidad.

Capítulo 3 Caso práctico

Para ilustrar el funcionamiento del sistema de IA de utilidad desarrollado, presentamos un ejemplo práctico que demuestra cómo los NPCs pueden tomar decisiones adaptativas y realistas en función de sus necesidades y el entorno. Este ejemplo ha sido diseñado para destacar la interacción de los distintos componentes del sistema, proporcionando una visión clara de su operación conjunta.

3.1 Configuración del entorno

En este ejemplo hemos configurado una escena en Unity con cuatro NPCs, tal y como se puede ver en la Figura 3.1:

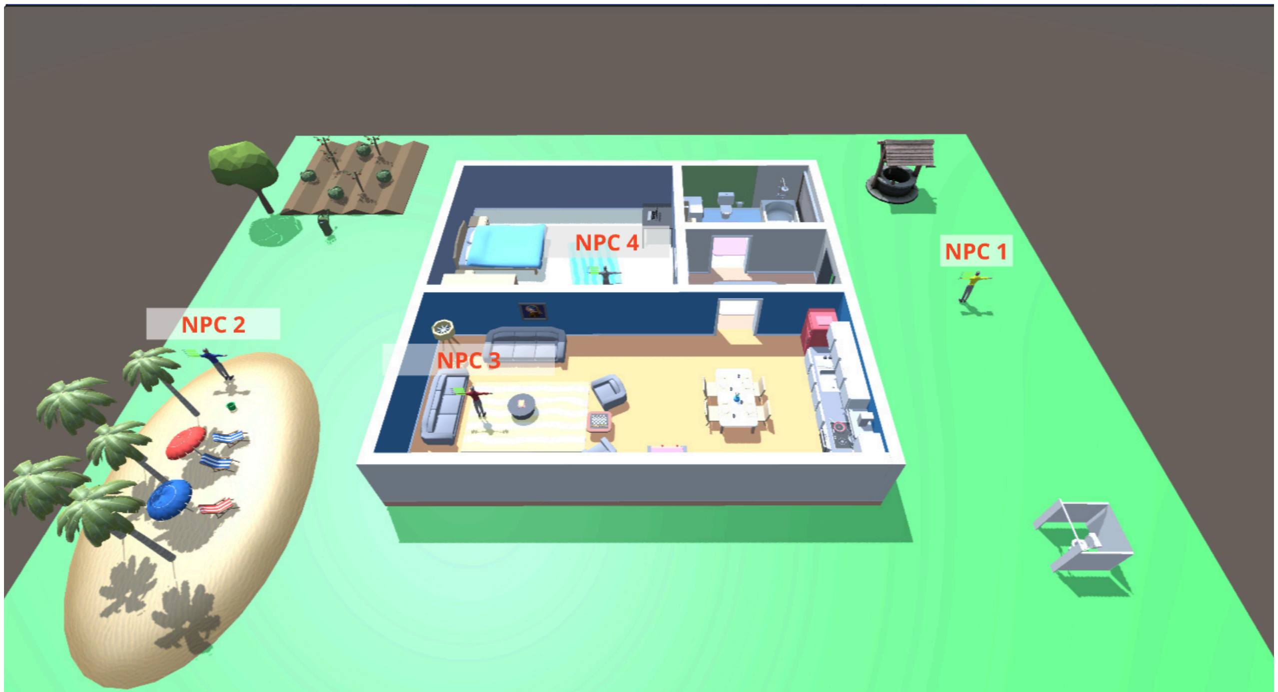


Figura 3.1: Escena de Unity.

A su vez, hemos definido las siguientes necesidades para cada NPC: hambre (hunger), energía (energy), diversión (fun), higiene (hygiene) y vejiga (bladder). Además, hemos considerado qué necesidades, como energía o hambre, tienen prioridad frente a otras, como diversión. Por esta razón, hemos definido las curvas de evaluación de las necesidad más importantes como exponenciales, mientras que las de las necesidades menos críticas son lineales.

Asimismo, para evitar que una necesidad con una función lineal supere a una con una función exponencial, hemos limitado la curva de diversión para que solo llegue a 0.5 en su nivel de urgencia máxima. Esto asegura que, incluso en su nivel más bajo, la diversión no se vuelva más urgente que el hambre, lo cual sería ilógico dada la prioridad de las necesidades fisiológicas. En la Figura 3.2 puede verse un ejemplo con la comparación de las curvas para las necesidades de diversión y hambre.

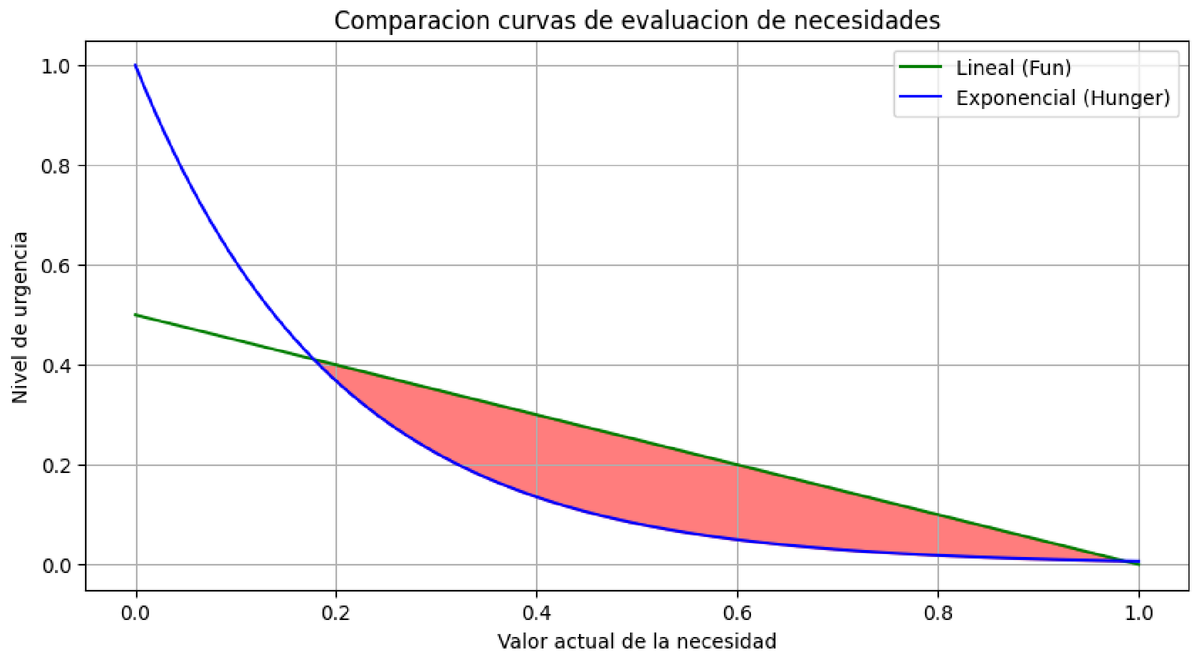


Figura 3.2: Comparación entre una curva lineal y exponencial.

Para esta simulación, todas las necesidades han empezado en su máximo valor (100). A continuación, en la Figura 3.3, podemos observar cómo son las curvas configuradas para todos los NPCs.

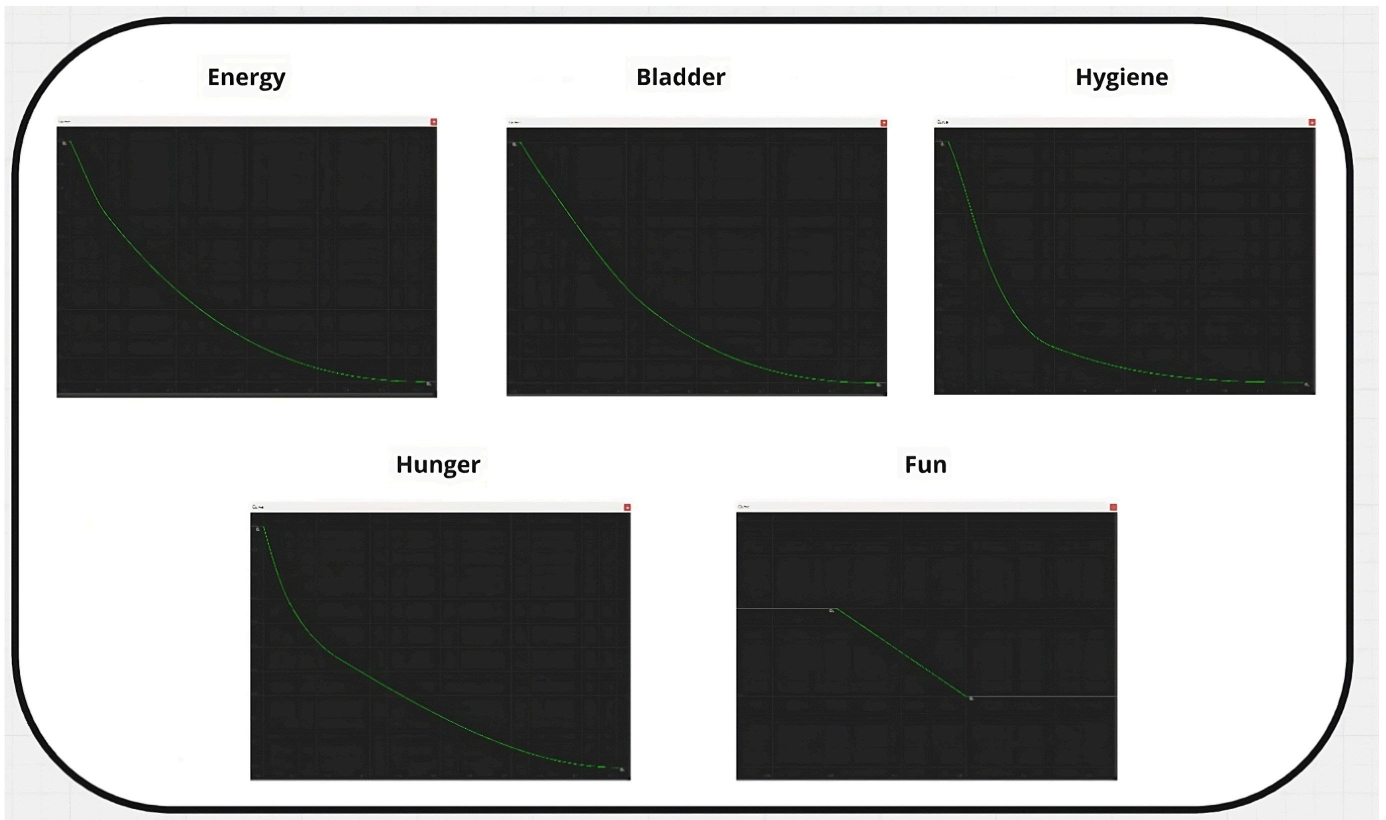


Figura 3.3: Curvas de las necesidades.

En la escena, hemos incorporado una variedad de ítems que los NPCs pueden utilizar para satisfacer sus necesidades. Estos ítems incluyen una cama (bed) para restaurar energía, una nevera (fridge) para satisfacer el hambre, un televisor (TV) para aumentar la diversión, una ducha (bath) para mejorar la higiene, o un inodoro (toilet) para aliviar la vejiga. A continuación, se presenta la Tabla 1 con los ítems disponibles en la escena, la necesidad que satisfacen, su valor inicial y su acción asociada:

Ítem	Necesidad que satisface	Valor inicial	Acción
Arcade	Fun	80	Have fun
Bath	Hygiene	100	Clean
Beach chair	Energy	20	Rest
Bed	Energy	100	Rest
Chess	Fun	30	Have fun
Dwell	Hygiene	70	Clean
Food	Hunger	20	Eat
Fridge	Hunger	80	Eat
Kitchen	Hunger	60	Eat
Laptop	Fun	30	Have fun
Mini sofa	Energy	30	Rest
Sofa	Energy	60	Rest
Toilet x2	Bladder	100	Empty bladder
Toilet sink	Hygiene	30	Clean
TV	Fun	60	Have fun
Watering can	Fun	40	Have fun

Tabla 1: Ítems de la escena.

Como se mencionó en el apartado de **2.3.3 Actions**, al ejecutar la acción asociada al ítem, el NPC se desplazará hasta él (teniendo en cuenta la distancia

configurada), usará el ítem (esperando la cantidad de segundos configurada) y tras ello volverá a evaluar su estado y considerará si debe ejecutar otra acción.

3.2 Ejecución y resultados

Para este ejemplo, se ha ejecutado la simulación durante un periodo de diez minutos. Tras ello, se han observado los siguientes comportamientos y resultados:

- **Interacción dinámica y adaptativa**: Los NPCs se movieron de manera coherente y realista hacia los ítems que satisfacían sus necesidades más urgentes. El uso de curvas exponenciales para necesidades críticas como hambre o energía permitió que estas tuvieran prioridad sobre otras menos importantes como diversión, tal y como se esperaba.

En la Figura 3.4 podemos observar un gráfico con la distribución de necesidades satisfechas para cada NPC.

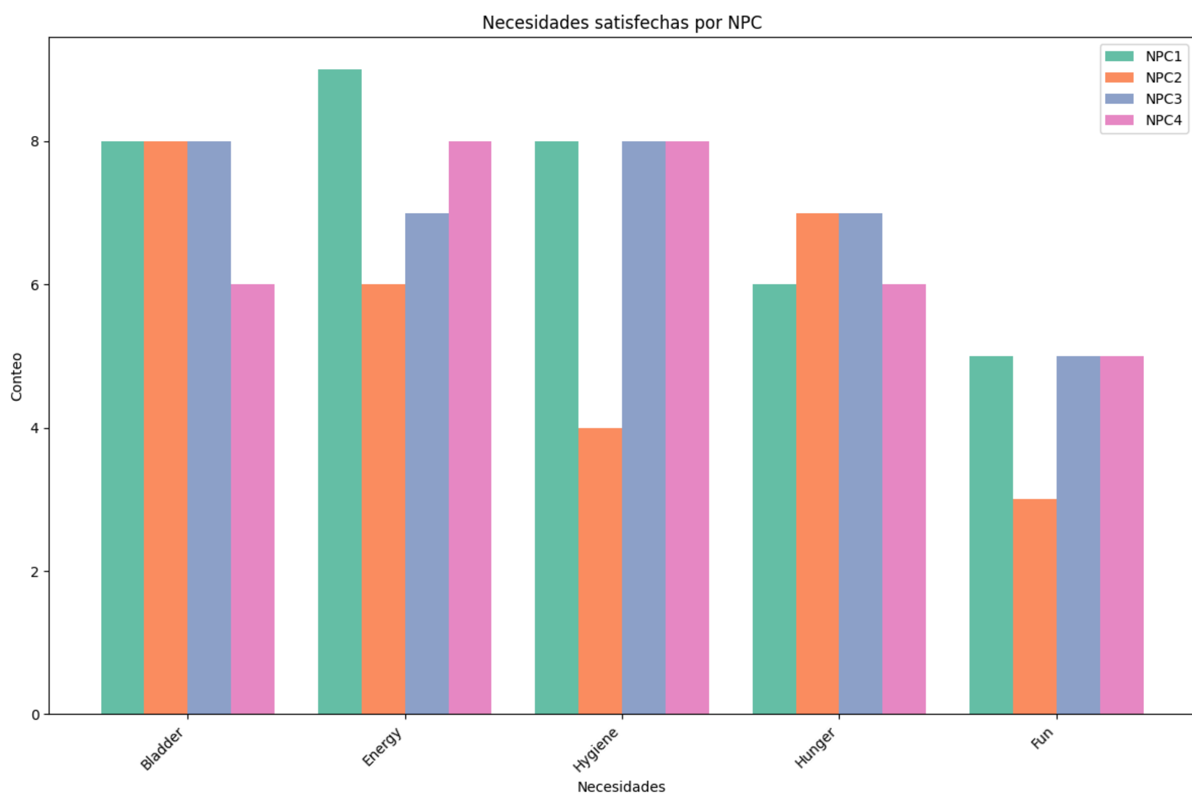


Figura 3.4: Distribución de necesidades para cada NPC.

- **Variedad en el comportamiento de los NPCs**: Debido a la selección aleatoria de los ítems entre los tres primeros con mayor puntuación, los NPCs mostraron comportamientos variados y no repetitivos.

La disminución aleatoria del valor de los ítems utilizados fomentó una rotación en el uso de diferentes ítems para satisfacer las mismas necesidades. Cabe mencionar que cuando un NPC ejecuta la acción de “Idle” significa que durante ese bucle de evaluación no hubo ninguna necesidad que satisfacer y por lo tanto el NPC se mantuvo en donde estaba.

A continuación, en las Figuras 3.5 y 3.6 podemos ver cómo se distribuyeron el uso de ítems - acciones respectivamente.

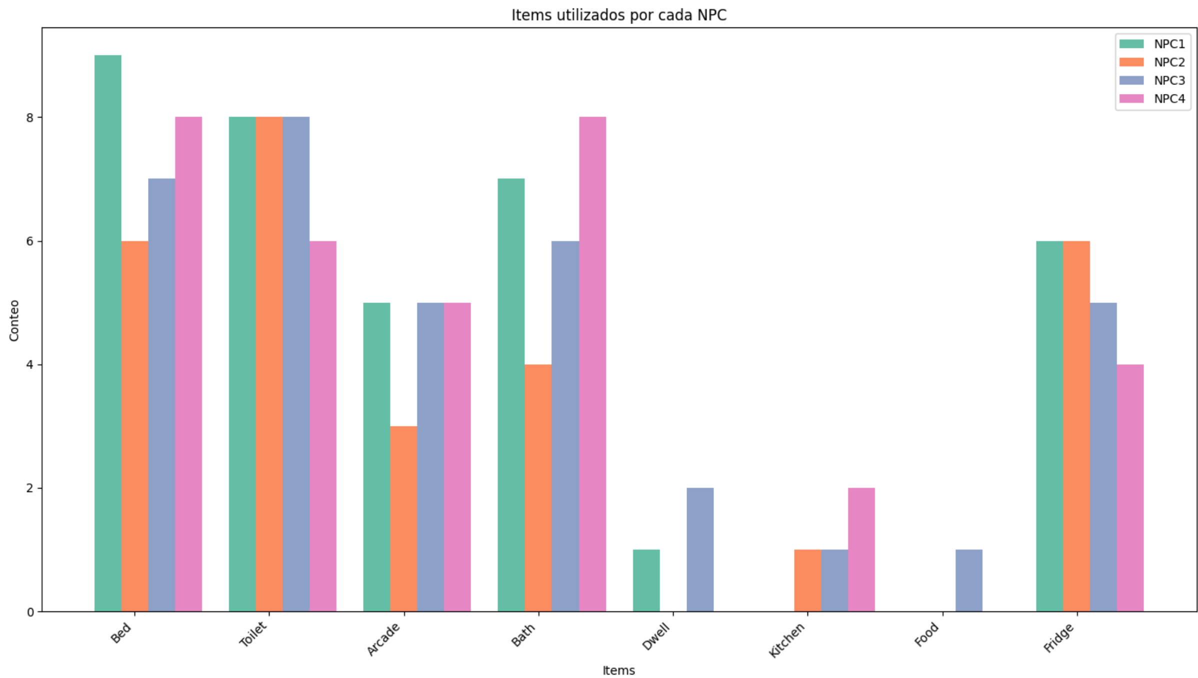


Figura 3.5: Distribución de ítems para cada NPC.

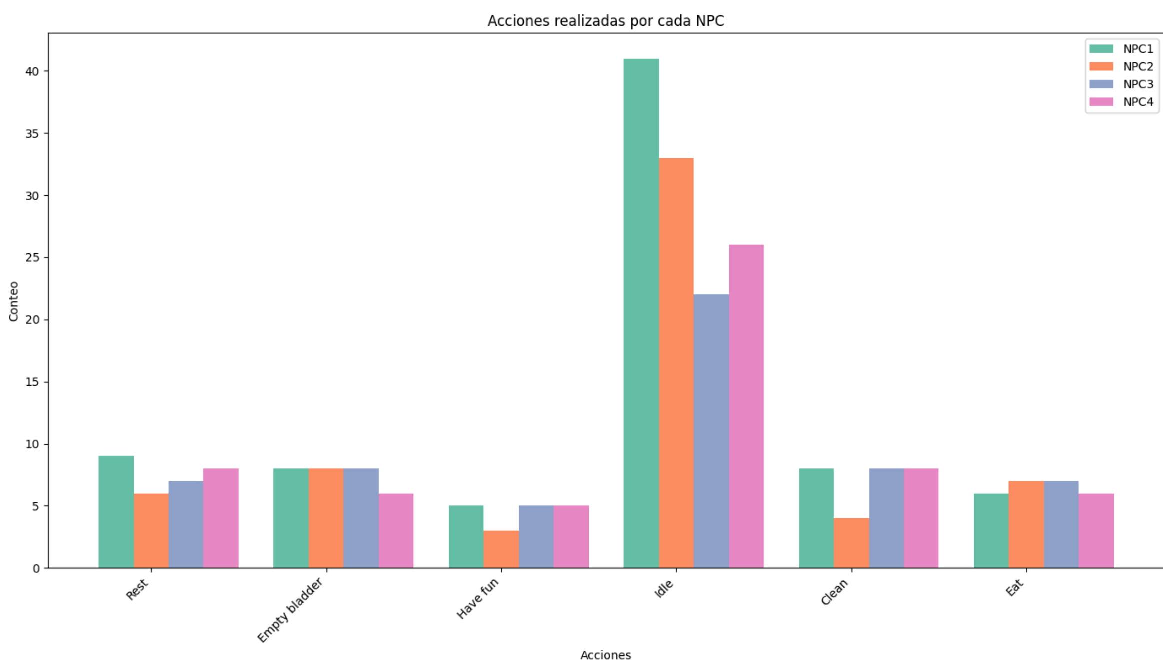


Figura 3.6: Distribución de acciones para cada NPC.

- **Feedback visual claro**: La generación de burbujas de texto y la actualización de *DataWindow* proporcionaron una retroalimentación clara y comprensible sobre el estado y las acciones de los NPCs.

Capítulo 4 Conclusiones y futuras líneas de trabajo

En este trabajo de fin de grado, hemos desarrollado un sistema de IA de utilidad en Unity, inspirado en la saga de videojuegos The Sims. Este sistema permite a los NPCs tomar decisiones adaptativas y realistas basadas en sus necesidades y el entorno que los rodea.

Una característica clave de nuestro sistema es que los objetos del entorno (ítems) reportan un valor específico para cada necesidad. Estos valores son ponderados en función del estado actual de cada motivo en el NPC, lo que permite que las decisiones sean contextualmente relevantes y adaptativas. Este enfoque se alinea con los principios observados en The Sims, donde los objetos "anuncian" su capacidad para satisfacer determinadas necesidades, y los NPC ejecutan las acciones basándose en la utilidad reportada por los objetos y su estado actual.

Para validar y demostrar el funcionamiento de nuestro sistema, hemos implementado un escenario en Unity. En este escenario, los NPCs interactúan con diversos objetos como camas, neveras o retretes, tomando decisiones que reflejan la prioridad de sus necesidades. Por ejemplo, un NPC con niveles bajos de energía priorizará el uso de una cama sobre otras actividades, mientras que un NPC con energía suficiente puede optar por el resto de acciones.

Finalmente, tras recopilar y analizar datos estadísticos de la simulación, hemos podido comprobar que el sistema se comporta de manera coherente con nuestras expectativas, dándose un reparto homogéneo de las diferentes acciones para mantener las necesidades de los NPCs cubiertas.

4.1 Futuras líneas de trabajo

A lo largo del desarrollo, se han enfrentado diversos desafíos, como la correcta implementación de las curvas de motivos y la sincronización de las animaciones con las acciones de los NPCs. Aunque se lograron resultados satisfactorios, aún existen áreas que podrían beneficiarse de futuras mejoras, por ejemplo:

- **Expansión de necesidades y acciones**: agregar nuevas necesidades y acciones para enriquecer el comportamiento de los NPCs. Además de poder

encadenar acciones.

- **Interacciones sociales**: implementar interacciones sociales entre NPCs para simular comportamientos más complejos y realistas.

- **Adición de animaciones**: añadir animaciones cuando se realizan acciones ya que ahora mismo solo se anima al NPC cuando este se desplaza.

- **Combinación con otras técnicas de IA**: explorar la posible combinación de este sistema junto con otras técnicas como GOAP o incluso técnicas de aprendizaje automático (Deep learning).

- **Manejo de NPCs**: como en The Sims, se podría añadir funcionalidad para manejar a los NPCs por parte del jugador y hacerlos ejecutar acciones.

- **Mejora de la interfaz**: enriquecer la información visual que se muestra como feedback de las acciones que se están planificando.

Chapter 5 Summary and future lines of work

In this final degree project, we have developed a utility AI system in Unity, inspired by the video game series The Sims. This system allows NPCs to make adaptive and realistic decisions based on their needs and the environment that surrounds them.

A key feature of our system is that environmental objects (items) report a specific value for each need. These values are weighted according to the current state of each motive in the NPC, allowing decisions to be contextually relevant and adaptive. This approach aligns with principles observed in The Sims, where objects "advertise" their ability to satisfy certain needs, and NPCs execute actions based on the objects' reported utility and current state.

To validate and demonstrate how our system works, we implemented a scenario in Unity. In this scenario, NPCs interact with various objects such as beds, refrigerators or toilets, making decisions that reflect the priority of their needs. For example, an NPC with low energy levels will prioritise the use of a bed over other activities, while an NPC with sufficient energy may opt for other activities.

Finally, after collecting and analysing statistical data from the simulation, we have been able to verify that the system behaves coherently with our expectations, with a homogeneous distribution of the different actions to keep the needs of the NPCs covered.

5.1 Future lines of work

Throughout the development, various challenges have been faced, such as the correct implementation of motive curves and the synchronisation of animations with NPC actions. Although satisfactory results were achieved, there are still areas that could benefit from future improvements, for example:

- **Expansion of motives and actions**: Adding new motives and actions to enrich NPC behaviour. Additionally, enabling the chaining of actions.

- **Social Interactions**: Implementing social interactions between NPCs to simulate more complex and realistic behaviours.

- **Addition of animations**: Adding animations when actions are performed since currently, we only have animations for NPC movement and the "idle" state.

- **Combination with other AI techniques**: Exploring the possible combination of this system with other techniques such as GOAP or even machine learning techniques (Deep Learning).

- **NPC management**: Similar to The Sims, functionality could be added to manage NPCs and make them perform actions.

- **Overall UI improvement**: all UI elements could be improved.

Capítulo 6 Presupuesto

Con respecto al presupuesto, podemos desglosarlo en el siguiente, tal y como se puede ver en la Tabla 2:

Trabajo	Tiempo (horas)	Coste (€/horas)	Coste total (€)
Busqueda de documentacion	20	20	400
Análisis de requisitos	20	20	400
Diseño de la librería	50	20	1.000
Desarrollo de prototipo	70	20	1.400
Búsqueda de assets (modelos y animaciones)	2	20	40

Desarrollo e implementación final	120	20	2.400
Implementación de casos de uso	10	20	200
Estudio de conclusiones	8	20	160
Total	**300**	X	**6000**

Tabla 2: Presupuesto.

Capítulo 7 Bibliografía

[1] «Non-player character», *Wikipedia*. 2 de julio de 2024. Accedido: 6 de julio de 2024. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Non-player_character&oldid=1232275870

[2] «Inteligencia artificial», *Wikipedia, la enciclopedia libre*. 8 de julio de 2024. Accedido: 10 de julio de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Inteligencia_artificial&oldid=161185197

[3] D. Aversa, «Chapter 1: Introduction to AI -- Finite state machines», en *Unity Artificial Intelligence Programming: Add Powerful, Believable, and Fun AI Entities in Your Game with the Power of Unity.*, 5th ed., Birmingham: Packt Publishing, 2022.

[4] M. Colledanchise y P. Ögren, «What are Behavior Trees?», en *Behavior Trees in Robotics and AI: An Introduction*, 2018, pp. 3-21. doi: 10.1201/9780429489105.

[5] G. I. Simari, «Chapter 2 Preliminary Concepts -- The BDI Model», en *Markov Decision Processes and the Belief-Desire-Intention Model Bridging the Gap for Autonomous Agents*, 1st ed. 2011., en SpringerBriefs in Computer Science. , New York, NY: Springer New York, 2011, p. 3. doi: 10.1007/978-1-4614-1472-8.

[6] J. Orkin, «Three States and a Plan: The A.I. of F.E.A.R.», presentado en Game Developers Conference, 2006.

[7] E. Arts, «Videojuegos de Los Sims: sitio oficial de EA», Electronic Arts Inc. Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://www.ea.com/es-es/games/the-sims

[8] «Dragon Age: Inquisition», *Wikipedia, la enciclopedia libre*. 24 de mayo de 2024. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Dragon_Age:_Inquisition&oldid=160320590

[9] N. Schoenholtz, «Three Kingdoms», Total War. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://www.totalwar.com/games/three-kingdoms/

[10] «Apex Utility AI - Personal Edition | Visual Scripting | Unity Asset Store».

Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://assetstore.unity.com/packages/tools/visual-scripting/apex-utility-ai-personal-edition-56306

[11] «Crystal AI». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://igiagkiozis.github.io/CrystalAI/

[12] J. Pennycook, «Pennycook/godot-utility-ai». 27 de junio de 2024. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://github.com/Pennycook/godot-utility-ai

[13] G. Engine, «Godot Engine - Free and open source 2D and 3D game engine», Godot Engine. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://godotengine.org/

[14] «Easy Utility AI in Code Plugins - UE Marketplace», Unreal Engine. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://www.unrealengine.com/marketplace/en-US/product/easy-utility-ai

[15] «La herramienta de creación 3D en tiempo real más potente», Unreal Engine. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://www.unrealengine.com/es-ES/home

[16] R. Zubek, «Needs-Based AI», en *Game Programming Gems 8*, 1st edition., Boston, MA: Cengage Learning PTR, 2010, pp. 302-311.

[17] M. Walkup, «AI Made Easy with Utility AI», Medium. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://medium.com/@morganwalkupdev/ai-made-easy-with-utility-ai-fef94cd36161

[18] *The Genius AI Behind The Sims*, (30 de junio de 2023). Accedido: 3 de julio de 2024. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=9gf2MT-IOsg

[19] «C# | Lenguaje de programación moderno y de código abierto para .NET», Microsoft. Accedido: 4 de julio de 2024. [En línea]. Disponible en: https://dotnet.microsoft.com/es-es/languages/csharp

[20] «Plataforma de desarrollo en tiempo real de Unity | Motor de 3D, 2D, VR y AR», Unity. Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://unity.com/

[21] «Visual Studio Code - Code Editing. Redefined». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://code.visualstudio.com/

[22] «Visual Studio: IDE y Editor de código para desarrolladores de software y Teams». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://visualstudio.microsoft.com/es/

[23] «Git». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://git-scm.com/

[24] «The most-comprehensive AI-powered DevSecOps platform». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://about.gitlab.com/

[25] «Pirámide de Maslow», *Wikipedia, la enciclopedia libre*. 10 de junio de 2024. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Pir%C3%A1mide_de_Maslow&oldid=160665469

[26] U. Technologies, «Unity - Manual: ScriptableObject». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://docs.unity3d.com/Manual/class-ScriptableObject.html

[27] U. Technologies, «Unity - Scripting API: NavMesh». Accedido: 4 de julio de 2024. [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/AI.NavMesh.html