SmartTech
LABS

# Reporte de IA-Simulación

Autores:
Claudia Alvarez Martínez-C412
Roger Moreno Gutiérrez-C412
Jan Carlos Pérez González-C412

## 1.0 Introducción

*SmartTech* es una compañía ficticia que se dedica a la creación de robots diseñados para servir como asistentes personales. El objetivo de este proyecto es proponer una manera de conocer la utilidad y los puntos a mejorar de este producto, en particular del último prototipo de la empresa: Will-E. Esto lo haremos a través de una simulación de las interacciones diarias entre la persona y el robot.

### 1.1 Para qué está concebido Will-E?

El objetivo fundamental de Will-E es servir y ayudar a la persona en las tareas diarias que pueden surgir en cualquier hogar. Está capacitado para cubrir diversas funciones, que pueden contribuir a la limpieza de la casa, el cuidado de las plantas, y ayudar a la persona en cualquiera de sus peticiones, contribuyendo a la satisfacción de sus necesidades, o entablando conversaciones de cualquier índole, gracias al modelo de lenguaje que tiene integrado este prototipo.

Will-E está equipado con un poderoso sistemas para recomendar recetas teniendo en cuenta diversas afectaciones de salud de la persona. Esto hace que se convierta en un asistente con funciones muy útiles para personas de avanzada edad, pues su espectro cubre enfermedades como la diabetes y la

hipertensión arterial, muy común en las personas mayores, proponiendo opciones que combinan los gustos de la persona, con el horario del día, y con los limitantes de una dieta que respeta y contribuye con el tratamiento de una enfermedad.

Según el IDF (International Diabetes Federation) aproximadamente 537 millones de personas padecen diabetes y 1300 millones de hipertensión arterial según fuentes WHO (World Health Organization).

Will-E cuenta con un conjunto de tareas de rutina, planificadas en los distintos días de la semana, entre las cuales se puede encontrar limpiar el suelo de las habitaciones de la casa, regar las plantas, o hacer una limpieza exhaustiva del baño. La empresa es consciente de que muchas más tareas de rutina se pueden programar en el robot.

## 1.2 Interrogantes que se planean responder

### 1.2.1 Rendimiento del robot

Esto se puede traducir en tiempo que demora haciendo tareas para las cuales está programado como rutina o bien las que surgen de su interacción con el humano (y también está programado para hacerlas). Ya que el robot puede interrumpir tareas por alguna razón y se quiere evaluar, entre otros detalles de rendimiento, las decisiones del robot.

### 1.2.2 Hora pico de solicitudes

Se trata de poder extraer las horas en las que el robot más contenido de trabajo tiene, lo cual la empresa puede usar, según los mercados donde se venda, para saber la hora que alcanza la temperatura máxima el robot y ver

si es factible en algunos mercados, o asimismo, para saber cuál es el mejor horario para el robot cargarse sin afectar en demasía la realización de sus tareas cotidianas. También, si el usuario trabaja o no, puede aprovechar para hacer sus tareas de rutina en ese tiempo de menos solicitudes.

### 1.2.3 Entendimiento de Lenguaje Natural

Se quiere conocer qué tan bien entiende las solicitudes del usuario el robot, sabiendo lo que le piden y lo que está recibiendo como mensaje podemos saber e intentar resolver posibles errores en la comprensión de lenguaje natural de Will-E.

## 2.0 Simulación

Primeramente, demos una caracterización, o vista general del medio y del comportamiento de nuestros agentes.

**La casa**: Representa el medio

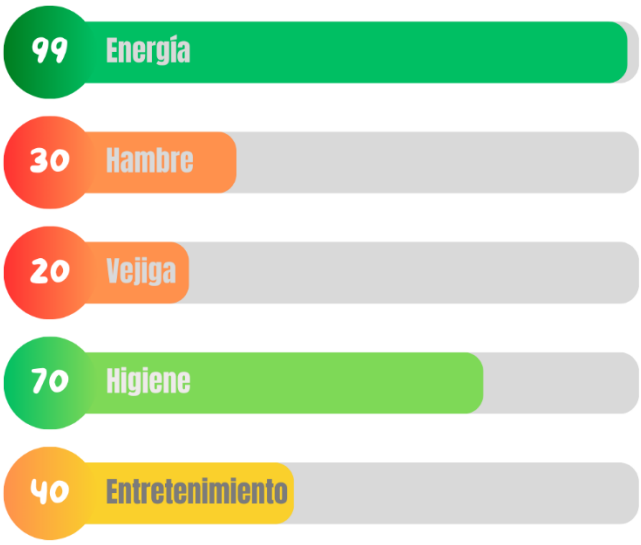
La casa va a ser nuestro escenario fundamental. Para simular la vida de una persona en ella, se ha determinado un espacio físico, constituido por

baldosas, por el que los agentes pueden caminar, y se han colocado por toda la casa objetos con los que los agentes pueden interactuar. Estos objetos van a contar con un conjunto de propiedades que definen restricciones en cuanto a la interacción con ellos. Ejemplos de estas propiedades puede ser *`portable`*, que define si un objeto lo puede portar algún agente; o la propiedad *`face_tiles`*, que define el conjunto baldosas por las cuales el objeto es accesible; o *`cleaneable`*, que determina si sobre ese objeto se puede emplear una acción de limpieza.

En el ambiente también se captan las ondas sonoras cuando los agentes hablan, pudiendo estos percibir de aquí las conversaciones, o poder el humano percibir la música que está escuchando.

**Pedro**: Representa al humano

Buscamos simular el comportamiento de una persona en una casa lo mejor posible, para ello hemos provisto a la persona de un conjunto de necesidades que debe cubrir, para ello puede realizar cualquier tipo de acción. Las necesidades tienen un valor en el rango [1-100] que representa cuan cubierta se encuentra esa necesidad, además están provistas de un índice de descenso que hace que disminuyan con el tiempo, así como también pueden subir con acciones para satisfacer cada una de ellas. Cada necesidad tiene un límite que representa el valor a partir del cual, la necesidad es importante cubrirla.




*Necesidades de Pedro*

A lo largo de la simulación, Pedro crea planes para satisfacer sus necesidades. Si el nivel de la necesidad es inferior al límite, inmediatamente Pedro planifica algo para satisfacer la necesidad indicada. Ejemplo de plan elaborado por Pedro para satisfacer el Hambre:

- *'Comer algo rápido'.*

Si la necesidad no está por debajo del límite, Pedro tiene la opción de pedir a Will-E que le ayude a satisfacer esa necesidad. De esta forma se puede ver una comunicación constante entre los agentes por medio del lenguaje natural, que los lleva a combinarse para cumplir un objetivo, saciar cierta necesidad. Veamos un ejemplo donde pedro pide a Will-E ayuda para satisfacer el Entretenimiento:

- *[08:45:30] Pedro planifica >Ordenar a Will-E para que ayude a satisfacer el entretenimiento.*
- *[08:45:31] Will-E planifica >Encender la TV y poner Netflix*

- *[08:46:03] Pedro planifica >Ver Netflix*

Pedro también puede hablar a Will-E sobre un tema, cuando está a punto de hacer una acción, esto puede condicionar de cierta forma la acción. Ejemplo donde Pedro habla a Will-E justo antes de comenzar el plan *'Ver una película'*.

➡ *Oye Will-E, ¿qué película puedo ver esta tarde?*

➡ *Hola Pedro, te recomiendo la película "The Martian". Es una película de ciencia ficción muy interesante.*

➡ *Gracias Will-E, le echaré un vistazo.*

➡ *De nada, Pedro. Que la disfrutes.*

**Will-E**: Representa nuestro robot asistente.

El robot asistente tiene definidas un conjunto de capacidades bien definidas que puede hacer en el medio, o sobre los objetos de la casa. Entre ellas está la capacidad de moverse por la casa; la de coger objetos, así como soltarlos; la de echar agua sobre objetos; o limpiar un objeto o área específica.

Will-E cuenta con una batería, para la cual se simula el nivel de carga. El nivel de la batería decrece con el tiempo, además, las distintas tareas que cumple influyen en incrementar el índice de descenso de la misma. Por ejemplo, tareas como caminar o limpiar algo, no consumen la misma cantidad de batería que otras como hablar.

Durante el día, Will-E, escucha las solicitudes de Pedro e inmediatamente elabora un plan para cumplirlas. El plan puede ser interpretado como simplemente responder a Pedro con información, o implicar realizar acciones sobre el medio. Posteriormente se explicará con más detalle el proceso de confección de planes.

Para simular el paso del tiempo, hemos hecho una discretización del mismo, definiendo (*1 segundo*), como la unidad mínima de tiempo. Luego por cada segundo, se hace una iteración donde se llama a una función run() en los agentes, para que hagan lo correspondiente a ese segundo.

Para la simulación se modelaron dos agentes BDI, uno que representa al robot asistente (Will-E) y el otro que representa al humano (Pedro), se encuentran implementados en `src/api/agents`. El archivo `bdi_agent.py` contiene la implementación de una clase abstracta de agente BDI, y las implementaciones concretas están en `bot_agent.py` y `human_agent.py` respectivamente.

Todo agente se inicializa de la siguiente manera:

```
59  class BDI_Agent(ABC):
60      def __init__(self, beliefs):
61          self.agent_id = None
62          self.beliefs = beliefs
63          self.desires = None
64          self.intentions = None
```

Y en los casos particulares del agente que modela al robot y al humano es de la siguiente manera respectivamente:

```python
class Human_Agent(BDI_Agent):
    def __init__(self, house: House, other_beliefs:dict):
        self.agent_id = 'Pedro'

class Bot_Agent(BDI_Agent):
    def __init__(self, house: House, other_beliefs: dict):
        self.__house = house    # private attribute
        self.agent_id = 'Will-E'
        self.human_id = 'Pedro'
        self.llm = Gemini()
        self.pocket = []
        self.beliefs = Bot_Belief(house, other_beliefs) # initial beliefs
        self.desires = ["Ayudar al humano en todo lo que pueda, en el hogar"]
        self.intentions: list[Plan] = []
        self.current_datetime = None
        self.battery = Battery()
```

Aquí cabe resaltar algunas propiedades interesantes como:

`last_given_order`: Es la última orden que le dio el humano al robot, una orden tiene dos propiedades importantes, si fue dada por el humano para satisfacer una necesidad y el cuerpo de la orden como tal.

`needs`: Las necesidades están representadas por una instancia de la clase `Need`.

`last_order`: Representa para el robot la última orden que recibió, es del mismo tipo que `last_given_order` que ya fue explicada.

`last_notice`: Representa la última conversación que tuvo con el humano (el humano también posee esta propiedad como creencia).

### 2.1 Flujo de la simulación de los agentes

Cada agente tiene un flujo de simulación similar, vamos a centrarnos en el agente que representa a Will-E que tiene más complejidad, ya que es el que se desea evaluar realmente en esta simulación. Para ilustrar de cierta manera qué es lo que hace el agente en cada iteración mostraremos las

siguientes líneas de código y luego se explicarán de manera sintetizada cada una que requiera ser explicada. La sección corresponde a la función run() del agente, invocada en cada segundo de la simulación.

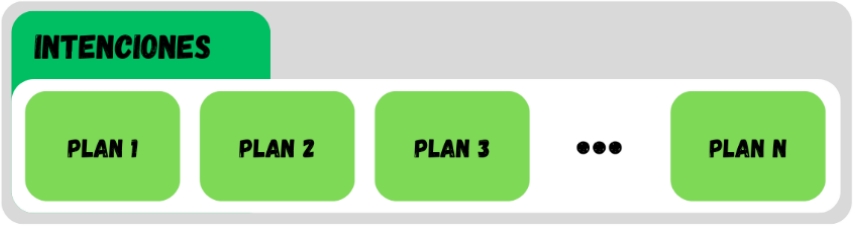
```
48    def run(self, submmit_event, current_datetime: datetime):
49        self.current_datetime = current_datetime
50
51        perception = self.see()
52        self.brf(perception)
53
54        self.plan_intentions()
55
56        if len(self.intentions) > 0:
57            current_plan: Plan = self.intentions[0]
58
59            if isinstance(current_plan, Charge):
60                self.battery.is_charging = True
61            else:
62                self.battery.is_charging = False
63
64            current_plan.run(submmit_event, current_datetime, self.beliefs.last_notice, self.battery)
65
66            if current_plan.is_successful:
67                self.intentions.pop(0)
68
69            self.increment_postponed_plan(current_plan)
70            perception = self.see()
71            self.brf(perception)
72
73            _reconsider, selected_intention = self.reconsider(current_plan, 0.1)
74
75            if _reconsider:
76                logger.log_overtake(current_plan, selected_intention)
77                self.reorder_intentions(selected_intention, current_plan)
78
79        else:
80            if not self.battery.is_charging:
81                self.battery.decrease_battery(0.0005)
```

Primeramente, el agente obtiene una percepción, a través de la función `self.see()`, esto lo que hace es tomar lo que pueda (que sea perceptible para él) del ambiente y luego actualiza sus creencias con la función `self.brf()`. Luego de esto planifica sus intenciones (se abordará más adelante como hace esto). Posteriormente ejecuta un segundo de simulación para el plan que actualmente se está llevando a cabo, eliminándolo de la lista de intenciones una vez este se complete. Luego de

que el robot actúe, este percibe nuevamente del entorno y actualiza sus creencias, ya que su acción puede haber significado un cambio en el ambiente. Después de cada acción los agentes deben reconsiderar, lo que se traduce en un reordenamiento de los planes en cola. En el caso de Will-E, si está en la ubicación donde tiene otra tarea que hacer, puede considerar llevarla a cabo, esto tiene un componente de aleatoriedad. Para el caso del robot por cada iteración de la simulación, aumenta el tiempo en que dejó tareas pospuestas (se argumentará en breve) y revisa el nivel de su batería.

### 2.1.1 Acerca de las tareas y los planes

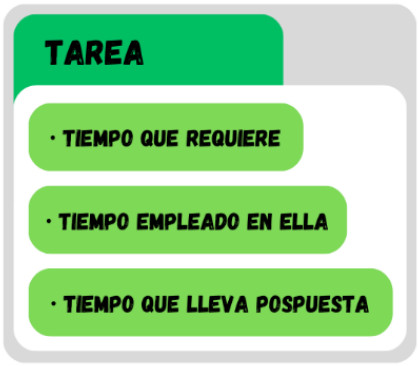
Los agentes cuentan con una cola de intenciones, a la que nos referimos anteriormente, donde cada intención es un plan que el agente ha confeccionado. Las intenciones son manejadas por él durante toda la simulación, pudiendo decidir el orden en que ejecuta los planes en cada segundo.




El plan en la cabeza de la lista de intenciones se encuentra el que será ejecutado, en cada segundo de la simulación. Un plan contiene una lista de tareas que el robot debe llevar a cabo para cumplirlo. Sobre el plan se guardan otras propiedades como por ejemplo, la de saber si el plan se

encuentra pospuesto, o la de saber si ese plan se lleva a cabo para satisfacer una necesidad del humano.


Una tarea cuenta con una función `execute()`, encargada de ejecutarse por cada segundo de la simulación, si es la tarea que en ese segundo se está realizando, como parte del plan actual. Existen diferentes tipos de tareas, que difieren fundamentalmente en la definición de la función `execute()`. De forma general cada tarea tiene, entre otras, las 3 propiedades mostradas a continuación.




Estas características permitirán analizar posteriormente el rendimiento del robot en el cumplimiento de tareas. Pues tareas que sean pospuestas, se pueden considerar en ocasiones como un mal rendimiento del robot.

El humano también cuenta con esa relación de Tarea-Plan, y funciona análogamente al robot, lo que las reconsideraciones no ocurren de la misma forma, ya que el humano solo ejecuta tareas para saciar sus necesidades, y mientras estas están saciadas, bien puede no hacer nada o lo que desee en ese momento (puede ser darle una orden al robot).

### 2.2 Procesamiento del lenguaje natural en la simulación

Uno de los usos fundamentales del lenguaje natural fue en la forma de comunicación entre los agentes, esto lo pudimos potenciar mediante el uso de modelos del lenguaje, para simular de forma más precisa la comunicación real. En este caso se usó una api de Gemini. Gemini se usa de una manera sencilla y las solicitudes se ejecutan con relativa velocidad en comparación con los demás LLM que había disponibles, y en este caso como se usa tan frecuente no se quiso prescindir de velocidad. Para el uso de Gemini se crearon más de 20 prompts, ubicados en el archivo **`prompts.py`**.

Se hizo un uso bastante extensivo del modelo del lenguaje, ya que se emplea desde simples conversaciones hasta elaboraciones de planes que debe llevar a cabo el agente, dicho de una manera más clara, al LLM se le dice que se desea hacer y cuáles son las acciones posibles y basado en esto elabora un plan, luego este “plan” es “parseado” para que pueda ser ejecutado por el agente. Las imprecisiones en la generación de planes las interpretamos como errores de interpretación del robot.

Otro uso notable, es que Will-E post procesa todas las conversaciones que tiene con el humano para identificar sus gustos y luego incluirlos en una base de conocimiento.

De forma general, el lenguaje natural es procesado constantemente durante la ejecución de toda la simulación.

## 2.3 Conocimiento y lógica difusa en los agentes

### 2.3.1 Recomendador de recetas

El agente Will-E posee varias funciones donde usa conocimiento, una de las más importantes, ya que es una de las tareas para las que fue concebido es la recomendación de recetas para personas con alguna enfermedad como hipertensión o diabetes, aunque también tiene soporte para dolientes de una gripe común, que, de cierta manera, puede requerir algunos platos especiales para una pronta recuperación. Todo esto se modeló con lógica difusa, aprovechando los conocimientos acerca de los padecimientos de la persona. En `recommenders.py` se ubica la implementación de los recomendadores, se usó para esto las librerías `skfuzzy` y `numpy`. Para conseguir el objetivo expuesto se dividen los recomendadores en dos, para más facilidad en la implementación; estos son: `RecipeRecommendationSystem` y `CulinaryStyleRecommendation` los cuales tiene los métodos `recommend_recipe` y `recommend_culinary_style`, ambos devuelven un mapeo del resultado de computar las reglas establecidas para cada objetivo, la primera sería cómo se debe comer la comida según las enfermedades (Ejemplo: Bajo de azúcar, bajo de sal, rico en vitamina C, etc)

y la segunda devuelve qué tipo de comida debe ser según la hora y los gustos del humano (Ejemplo: Almuerzo mediterráneo, Cena cubana, Desayuno mexicano). Ahora veamos las reglas que se usaron para el primer caso:

```
48      def _create_rules(self):
49          rule1 = ctrl.Rule((self.diabetes['high'] | self.diabetes['average']) & (self.heart_disease['high'] | self.heart_disease['average']) &
50                            (self.cold['high'] | self.cold['average']), self.recommended_dish['low_sugar_and_low_salt_and_C'])
51          rule2 = ctrl.Rule((self.diabetes['high'] | self.diabetes['average']) & self.heart_disease['low'] & self.cold['low'], self.recommended_dish['low_sugar'])
52          rule3 = ctrl.Rule(self.diabetes['low'] & (self.heart_disease['high'] | self.heart_disease['average']) & self.cold['low'], self.recommended_dish['low_salt'])
53          rule4 = ctrl.Rule(self.diabetes['low'] & self.heart_disease['low'] & (self.cold['high'] | self.cold['average']), self.recommended_dish['with_C'])
54          rule5 = ctrl.Rule((self.diabetes['high'] | self.diabetes['average']) & (self.heart_disease['high'] | self.heart_disease['average']) & self.cold['low'],
55                            self.recommended_dish['low_sugar_and_low_salt'])
56          rule6 = ctrl.Rule(self.diabetes['low'] & self.heart_disease['low'] & self.cold['low'], self.recommended_dish['normal'])
57          rule7 = ctrl.Rule(self.diabetes['low'] & (self.heart_disease['high'] | self.heart_disease['average']) & (self.cold['average'] | self.cold['high']),
58                            self.recommended_dish['low_salt_and_C'])
59          rule8 = ctrl.Rule((self.diabetes['average'] | self.diabetes['high']) & self.heart_disease['low'] & (self.cold['average'] | self.cold['high']),
60          self.recommended_dish['low_sugar_and_C'])
61
62          self.recipe_recommendation_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8])
63          self.recipe_recommendation = ctrl.ControlSystemSimulation(self.recipe_recommendation_ctrl)
```

Para cada una de los parámetros se establecieron rangos que representan bajo, medio y alto nivel de lo que sea el parámetro. Luego de computar esas reglas se mapea la salida a algo de lo que se dijo antes.

Análogamente a lo mostrado anteriormente se hace con el otro recomendador (mucha densidad de código como para mostrar una imagen).

### 2.3.2 Análisis para cargar la batería

Otro punto interesante, donde se usa conocimiento, o más bien, se toman decisiones informadas, es para decidir el mejor horario de carga del robot. Por defecto, siempre que el porcentaje de batería del robot descienda del 20%, se planifica ir a la estación a cargar. El robot también puede planificar cargas preventivas, y esto es, que cuando hace un nuevo plan, si el

consumo de batería estimado de ese plan hace que el nivel de batería descienda del 10%, se planifica cargar antes.

Diariamente, Will-E hace un registro de los minutos del día en que se detecta actividad. En base a este conocimiento se determina el intervalo del día, donde menos actividad tiene Will-E, estableciendo así la hora de carga óptima. Para determinar dicho intervalo, se toma una ventana de 4 horas, donde la suma del promedio de actividad por cada minuto del día, por todos los días anteriores, sea mínima.

### 2.4 Búsqueda en la simulación

Los agentes pueden caminar por la casa, la casa, está formada por baldosas, donde cada una puede tener objetos encima y estos pueden tener diferentes propiedades. La implementación de la solución de este problema se encuentra en `search.py` y se explica a continuación:

Se ofrecen herramientas para abordar problemas donde se necesita encontrar una secuencia de acciones óptima para ir de un estado inicial a un estado objetivo.

Se centra en el concepto de nodos, que representan posibles estados en el proceso de búsqueda. Cada nodo tiene información sobre el estado actual, el nodo padre que lo precede en la búsqueda, la acción que llevó a ese estado y el costo del camino desde el estado inicial hasta ese nodo.

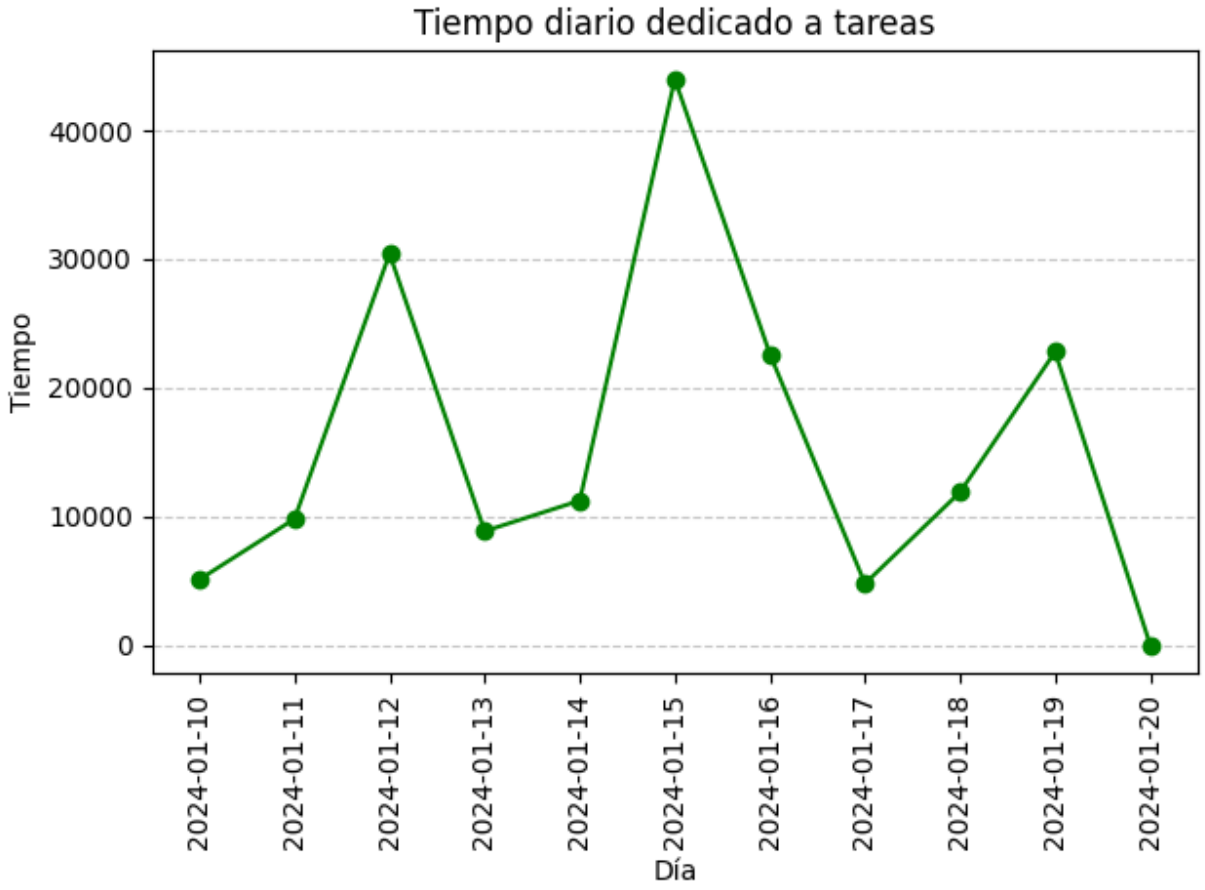
La estructura define métodos para expandir un nodo dado, generando sus nodos hijos en función de las acciones posibles disponibles en ese estado. Además, proporciona funciones para obtener la secuencia de acciones y estados desde el nodo inicial hasta un nodo dado.

Se definió el problema de caminar de una baldosa a otra, utilizando en cuanto a algoritmo de búsqueda, la búsqueda A*, complementado con la función heurística de Manhattan.

### 3.0 Resultados

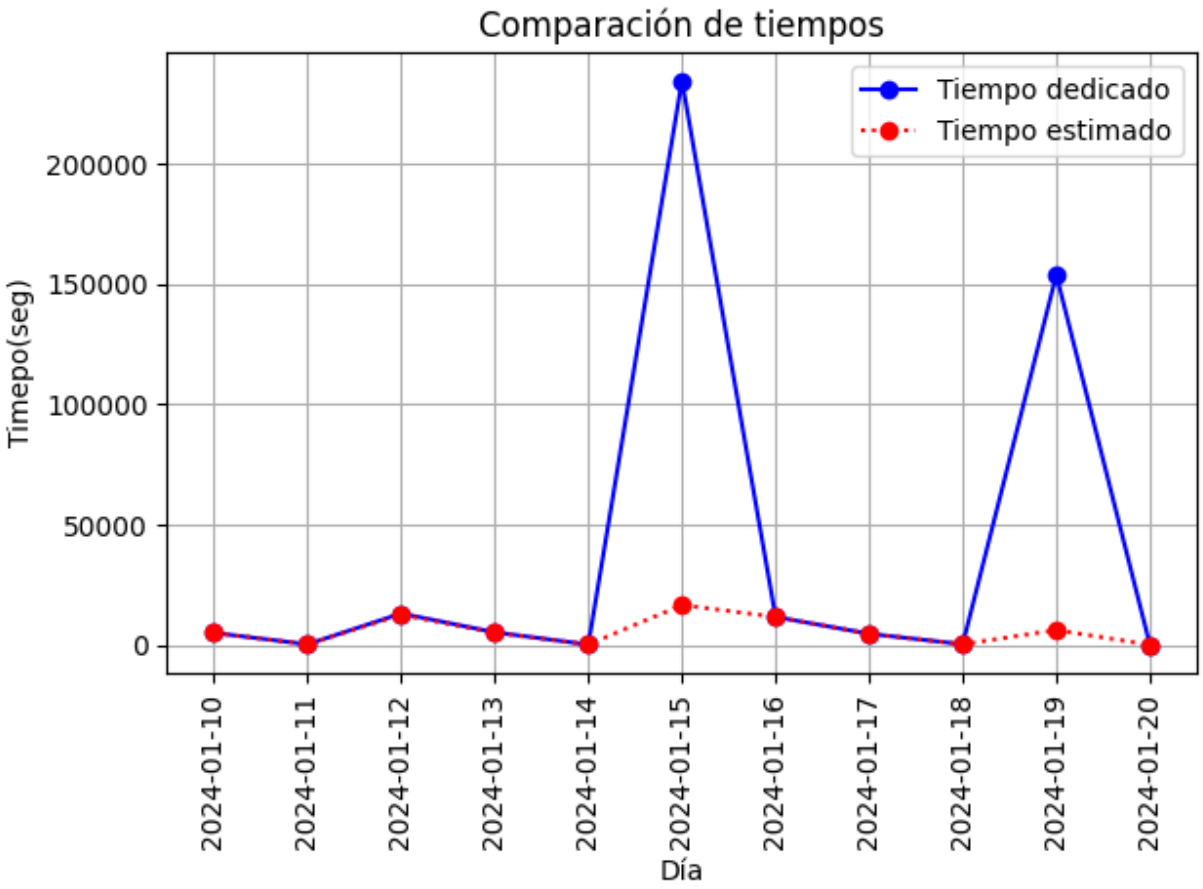
Para evaluar los resultados, se hizo una simulación de 7 días, de donde se extrajeron algunas estadísticas relacionadas con los tópicos que se abordaron en la sección 1.2. Comenzando por la evaluación del rendimiento del robot, se tiene lo siguiente:

### 3.1 Rendimiento del robot



Tenemos en el gráfico anterior el tiempo total en segundo dedicado a tareas realizadas por el robot en cada día de la simulación. Se puede notar como los días que el robot tiene asignado la limpieza de la casa son los días que más tiempo trabajado tiene, y que el día que más trabajó fue el 15 de enero de 2024, que es lunes, día de la limpieza exhaustiva de la casa, trabajó 40000 segundos, equivalente a 11.11 horas, en total trabajó en la semana 171440 segundos (47.62 horas), para una media de 15585.4545 segundos (4.32 horas).

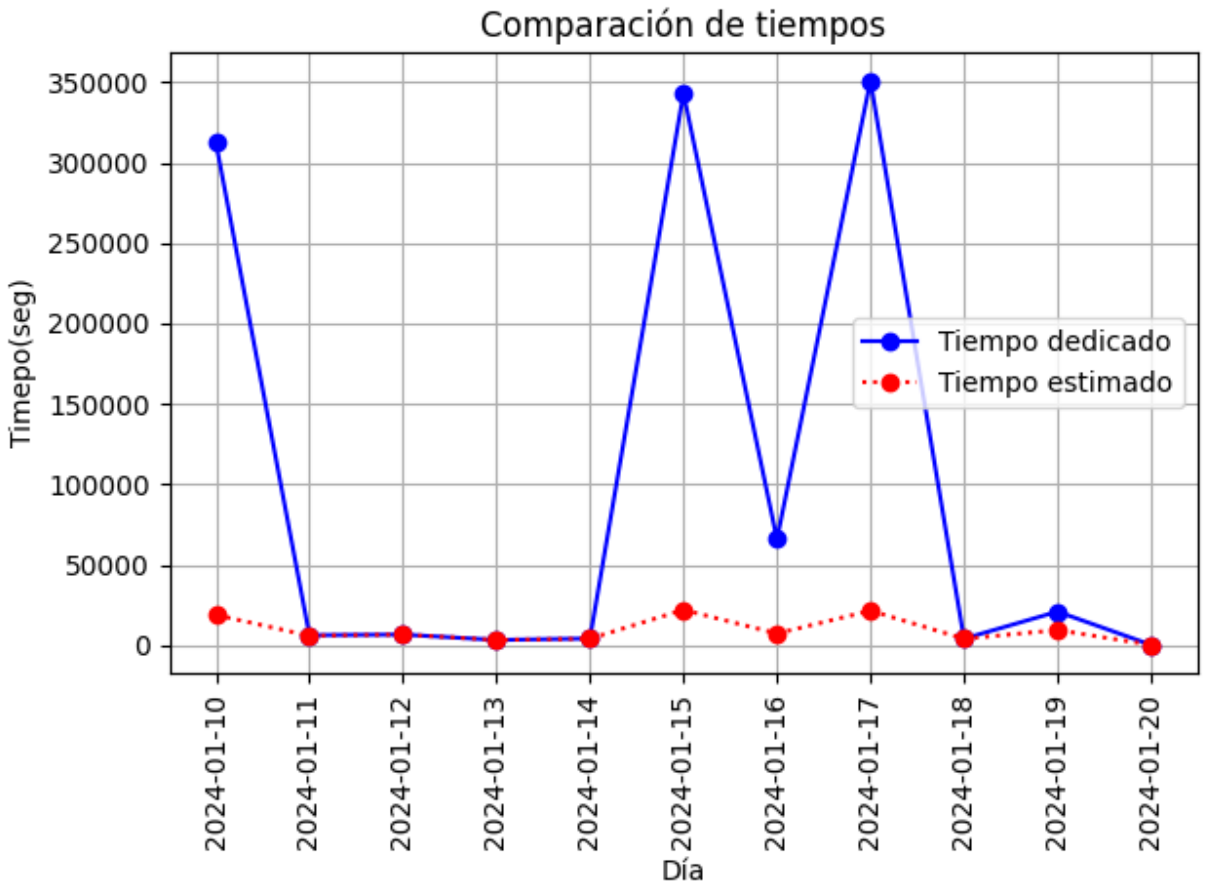
Ahora analizaremos cómo afectó a su rendimiento las tareas pospuestas por Will-E, para ello nos apoyaremos del siguiente gráfico que muestra dos líneas, una para el tiempo que debió haber demorado realizando una tarea (según sus cálculos)  y otra para el tiempo que realmente demoró (desde que la empezó a hacer hasta que la terminó), de aquí cabe destacar que lo que se muestra es por día la suma de los dos tiempos que se explicaron anteriormente, notar que el día que peor rendimiento tuvo fue el día que más tareas tuvo que realizar ya que coincidía con la jornada de limpieza de la casa. Está claro que el día que más tareas realizó fue el que más alto amplia tuvo la diferencia de los tiempos.




Para complementar esta información se muestra la siguiente tabla con la misma información y además el porcentaje de eficiencia por cada uno de los días (filas), para un rendimiento promedio de 82.13%.

| | Tiempo estimado | Tiempo total | Tiempo pospuesto | % de eficiencia |
|---|---|---|---|---|
| 0 | 4967.0 | 4967.0 | 0.0 | 100.0 |
| 1 | 243.0 | 243.0 | 0.0 | 100.0 |
| 2 | 12489.0 | 13064.0 | 575.0 | 95.59859154929580 |
| 3 | 5072.0 | 5218.0 | 146.0 | 97.20199310080490 |
| 4 | 131.0 | 131.0 | 0.0 | 100.0 |
| 5 | 16563.0 | 234804.0 | 218241.0 | 7.053968416210970 |
| 6 | 11622.0 | 11622.0 | 0.0 | 100.0 |
| 7 | 4597.0 | 4616.0 | 19.0 | 99.58838821490470 |
| 8 | 268.0 | 268.0 | 0.0 | 100.0 |
| 9 | 6168.0 | 153993.0 | 147825.0 | 4.005376867779700 |
| 10 | 12.0 | 12.0 | 0.0 | 100.0 |

Ahora se mostrará además cómo se comporta Will-E en una simulación donde se le asignó muchas más tareas algunos días y además se puso en 0.9 la probabilidad que se explicó antes en la sección 2.1.1 respecto a con qué frecuencia reconsideraba su plan actual según la ubicación donde estaba ahora mismo Will-E, para ello mostraremos esta misma gráfica y tabla, ahora correspondientes a la simulación antes descrita.




Por cuestiones de espacio no se puede mostrar una tabla con las tareas que fueron pospuestas por la razón antes descrita, y cuántas de las

pospuestas eran para satisfacer una necesidad de Pedro, lo que se puede traducir en una fallo de rendimiento claro, por tanto se adjunta a este reporte dicha tabla que igualmente será descrita aquí.

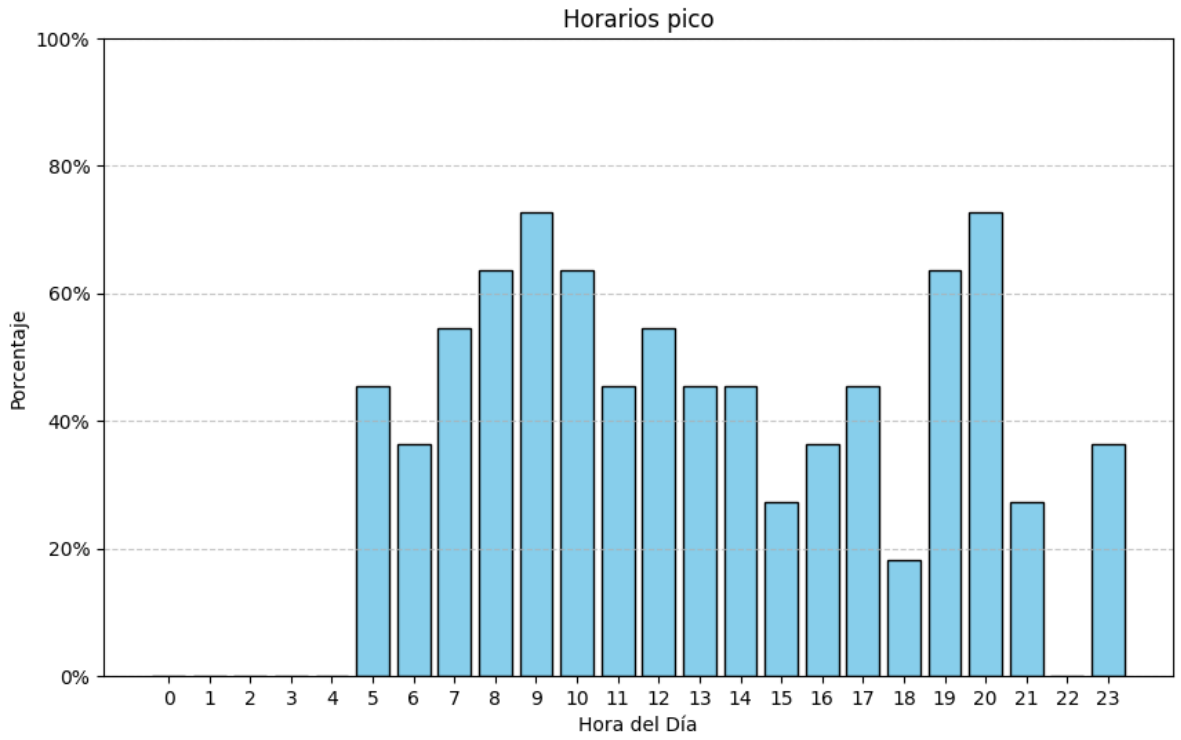
| | Tiempo estimado | Tiempo total | Tiempo pospuesto | % de eficiencia |
|---|---|---|---|---|
| 0 | 18764.0 | 312431.0 | 293667.0 | 6.005806081982900 |
| 1 | 5842.0 | 6148.0 | 306.0 | 95.02277163305140 |
| 2 | 6562.0 | 6562.0 | 0.0 | 100.0 |
| 3 | 3199.0 | 3199.0 | 0.0 | 100.0 |
| 4 | 3977.0 | 4105.0 | 128.0 | 96.88185140073080 |
| 5 | 21932.0 | 343164.0 | 321232.0 | 6.39111328694152 |
| 6 | 7200.0 | 65848.0 | 58648.0 | 10.934272870854100 |
| 7 | 21414.0 | 351078.0 | 329664.0 | 6.099499256575460 |
| 8 | 3971.0 | 3987.0 | 16.0 | 99.59869576122400 |
| 9 | 9343.0 | 20833.0 | 11490.0 | 44.847117553880900 |
| 10 | 166.0 | 166.0 | 0.0 | 100.0 |

Podemos notar en esta simulación que el rendimiento promedio disminuye muchísimo, siendo ahora de 60.52%.

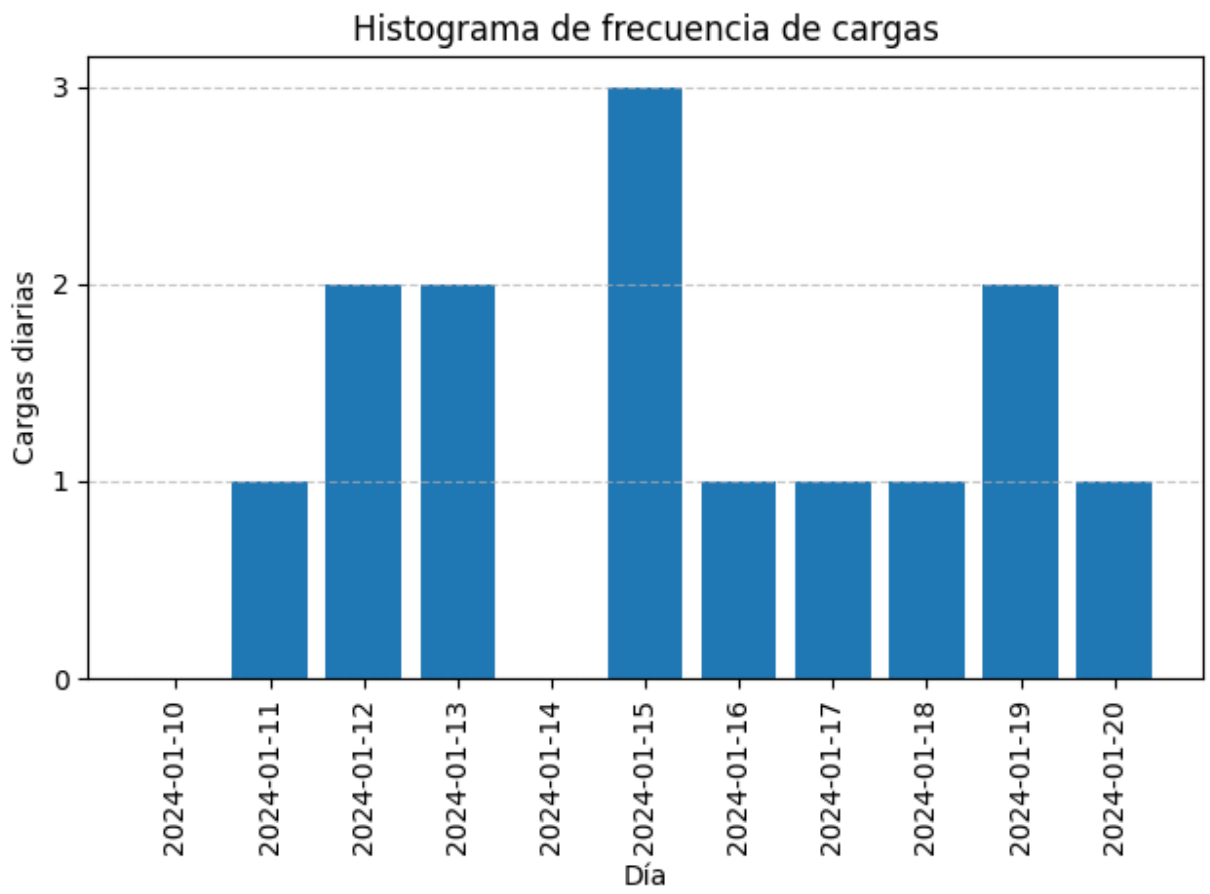
La otra tabla de la que se habló, posee 27 filas, y de ellas absolutamente todas eran tareas de necesidad para Pedro y se cambiaron por tareas que no eran de necesidad, lo que muestra una clara ineficiencia y nos deja como aprendizaje que este tipo de reconsideración debe estar equilibrada, usualmente (en las demás simulaciones) no pasaba esto porque Will-E no tenía la suficiente cantidad de tareas diarias como para que se saturara de esta manera y coincidiera tantas veces que mientras estaba en una habitación tenía algo que hacer en ella.

### 3.2 Horas pico de solicitudes

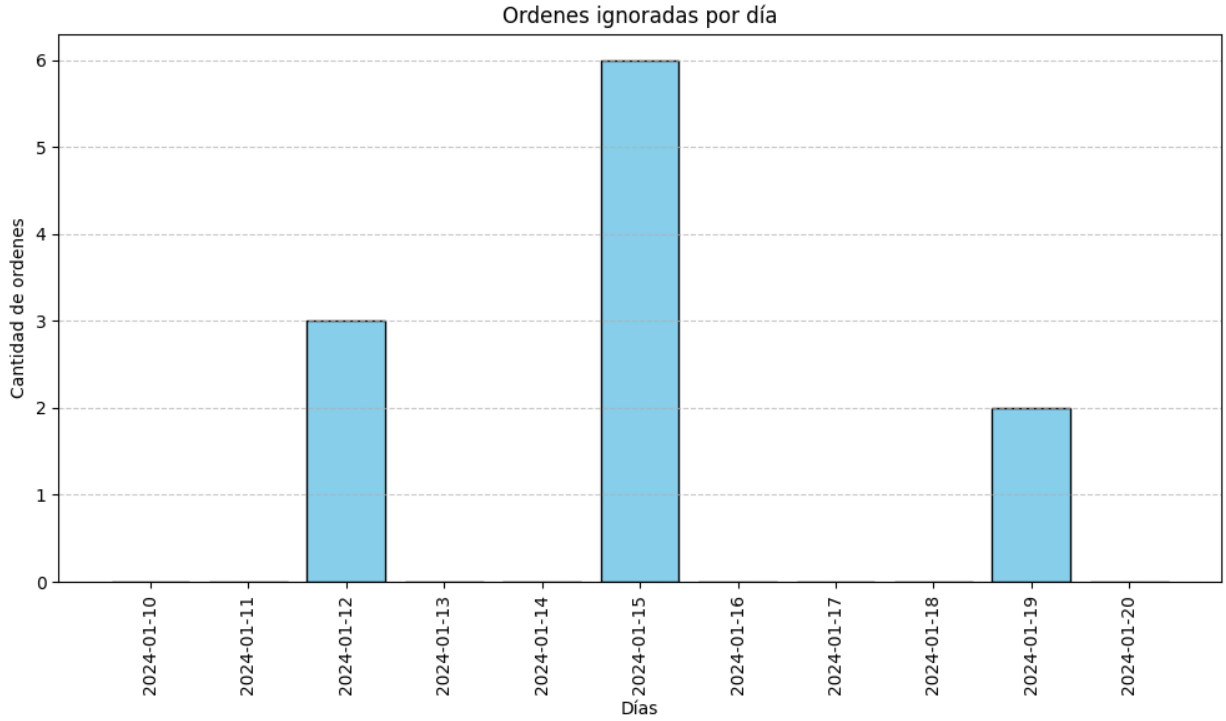
Como se dijo anteriormente las horas pico de solicitudes son importantes para conocer a qué hora se debe cargar Will-E, por esto se recoge la estadística de cuántas veces se carga por día, cuantas órdenes deja de procesar por estar cargando y cómo evoluciona el tiempo de carga de acuerdo al conocimiento que tiene el robot sobre el día del usuario. Para ilustrar esto se muestran los siguientes gráficos:



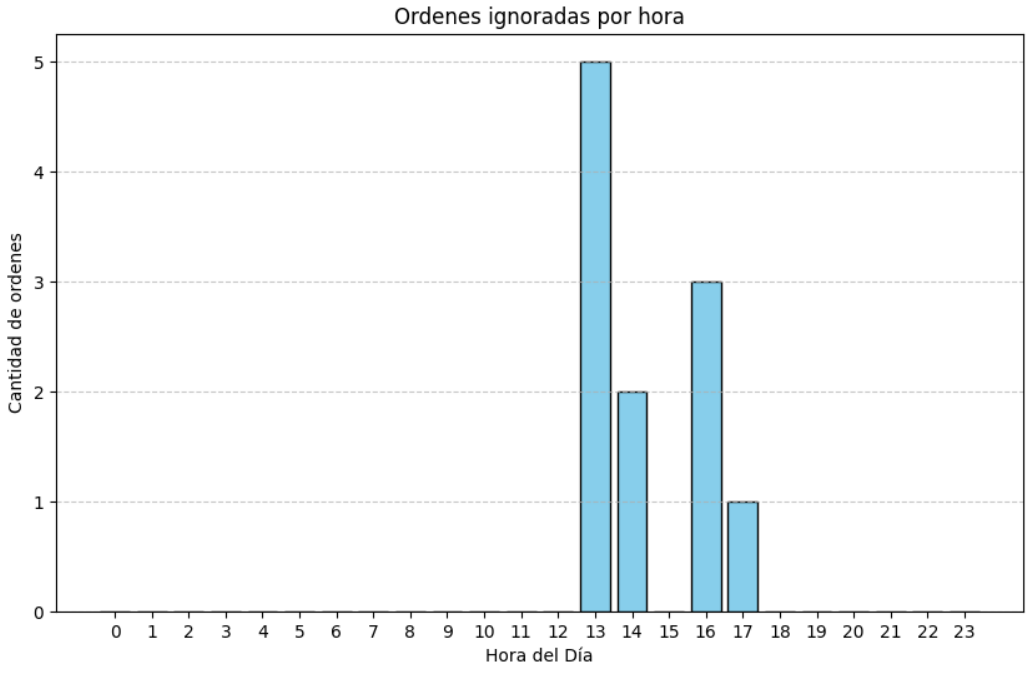

Aquí se muestran los horarios con el porcentaje de días en el que a esa hora Will-E hacía algo. Como se podría suponer la hora de la madrugada es la que menos frecuencia tiene porque es la hora de sueño de Pedro (cargarse no se considera actividad).



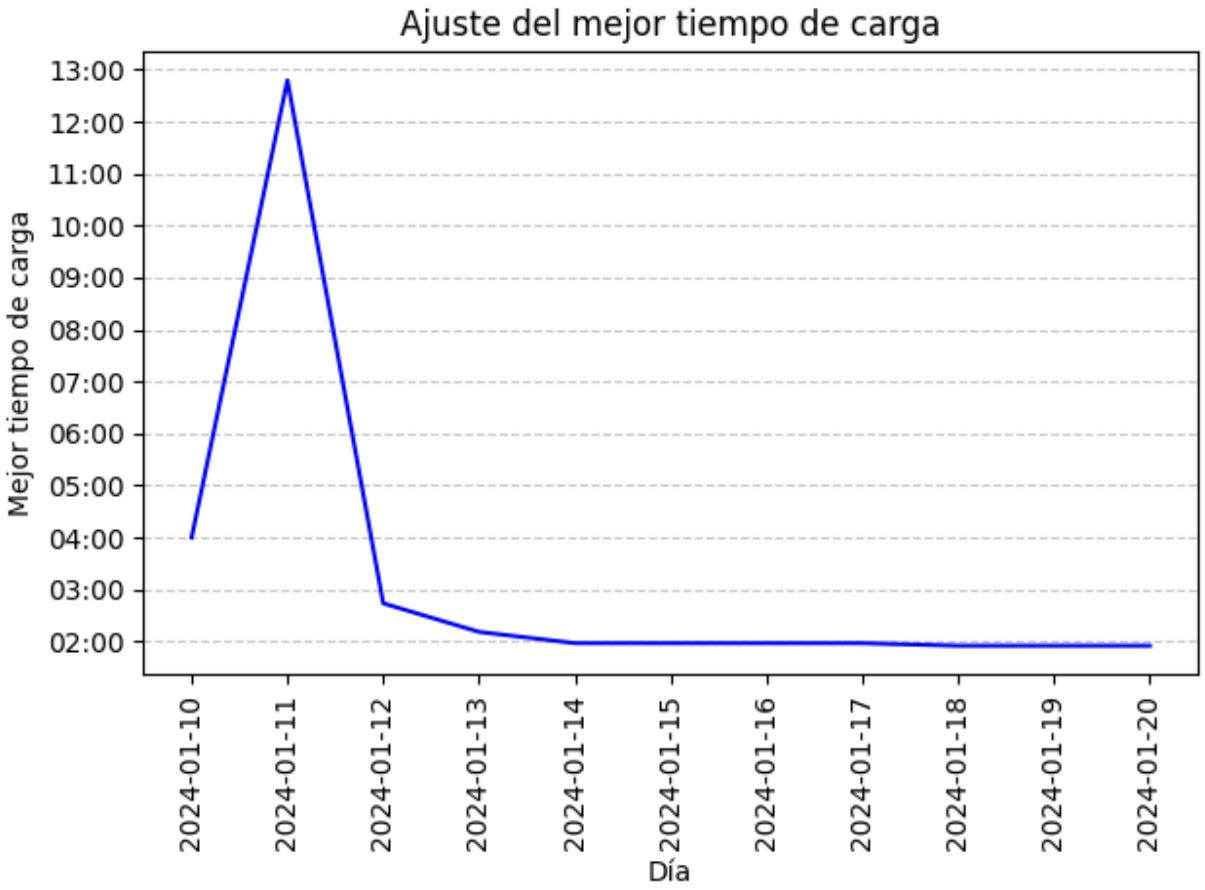

El histograma de frecuencia de cargas muestra que el día que más tiempo realizó tareas fue el que requirió que más veces se cargara (15 de enero), lo que puede afectar su rendimiento de cierta manera, ya que Will-E cuando carga no puede procesar órdenes, lo cual se mostrará a continuación:




Aquí es obvio que el día que más ordenes se ignoraron fue el día que más se cargó, veamos ahora un histograma de las ordenes ignoradas por hora, donde si abrimos los logs de la simulación nos daremos cuenta que está este log específico: *2024-01-15 12:57:18,Cargar la batería,0.0,9744.0,0.0,* lo que nos da una idea de porqué las 13 es la hora con más ordenes ignoradas.



En el gráfico siguiente vemos cómo se comportó la curva de adaptación al horario para cargarse, aquí se puede notar como comienza a una hora por defecto y al pasar de los días va convergiendo a las horas donde no hay actividad, notamos que el segundo día no hubo actividad a las 13 y se cargó a esa hora, pero lo normal es que se cargue a las horas que comenzó a hacerlo después ya que coincide con las horas sin actividad ninguna.




También Will-E tiene una función que se llama Carga preventiva, que es básicamente analizar antes de ejecutar una tarea si tiene carga para hacerlo; veamos ahora la tabla de las cargas preventivas para esta simulación:

Cargas preventivas

| Fecha | Nombre de la intención | Necesidad |
|---|---|---|
| 2024-01-15 09:37:01 | Realizar limpieza completa a la sala de estar | false |
| 2024-01-15 12:57:17 | Realizar limpieza completa a la cocina | False |

Se puede notar que las tareas no eran simples y requerían tiempo, por lo que tiene sentido.

### 3.3 Entendimiento de lenguaje natural

Gran parte del entendimiento de Will-E, a las órdenes de Pedro, pasa por el entendimiento del modelo del lenguaje. Esto va desde interpretar la intención de la acción que se quiere realizar, hasta establecer una secuencia de tareas lógicas, basadas en las limitadas acciones y objetos con los que interactuar.

Como parte de la simulación, se identifican órdenes de Pedro que Will-E no supo identificar o planificar. Por ejemplo:

- Pedro dice: Oye Will-E, prepara el baño con agua y jabón.
- Pedro dice: Oye Will-E, prepara algo de comer en el microondas.

Parte de el no entendimiento pasa por intentar planificar acciones sobre objetos que no existen en la casa, como es el caso del jabón o el microondas.

Ocasionalmente Will-E no logra crear planes con cierta complejidad, o que combinan muchas acciones, por ejemplo:

- Pedro dice: Oye Will-E, prepara el café y tráemelo a la sala de estar.

- "Pedro dice: Oye Will-E, prepara un café y llévalo a la mesa del comedor."
- "Pedro dice: Oye Will-E, prepara un baño bien calentito con espuma para mí."
- "Pedro dice: Oye Will-E, prepara un baño con espuma caliente"
- "Pedro dice: Oye Will-E, prepara un baño con espuma y toallas calientes."

Podemos observar que el caso de preparación de baños calientes no lo pudo reconocer en la mayoría de las ocasiones.

Otro ejemplo de no entendimiento que fue recurrente, fue en los casos en que Pedro preguntaba la hora a Will-E, a pesar de este tener conocimiento de la hora de la simulación en todo momento.

- "Pedro dice: Oye Will-E, ¿me recuerdas que hora es mientras voy al baño?"

## 4.0 Conclusiones

El análisis detallado de la simulación de Will-E proporciona una base sólida para realizar ajustes significativos en su diseño y funcionamiento. El tiempo promedio dedicado a las tareas diarias, el impacto de las tareas pospuestas en el rendimiento y la identificación de horas pico de solicitudes son datos valiosos que SmartTech puede utilizar para optimizar la eficiencia de Will-E.

En resumen, el estudio mostró que:

i) El día con mayor carga de trabajo fue el día de limpieza de la casa, afectando significativamente el rendimiento debido a las tareas pospuestas.

ii) El robot aprende y adapta su horario de carga a las horas de menor actividad, mejorando su disponibilidad para procesar órdenes.

iii) La identificación de errores en el reconocimiento del lenguaje natural ayudará a mejorar la interacción humano-robot, aumentando la satisfacción del usuario.

Estos insights permiten a SmartTech realizar ajustes en las áreas críticas de gestión de tareas y reconocimiento de lenguaje natural, asegurando que Will-E se convierta en un asistente personal más eficiente y útil.

## 4.1 Aspectos a mejorar

Se considera que para conseguir mejores resultados en la simulación se podría haber complejizado (haber hecho más realista) el rango de visión del producto, ya que se asumió que el Will-E sabe en todo momento donde está un objeto, lo cual sabemos que no es cierto. Para ello se hubiera simulado la

percepción del robot mediante sensores lo que solucionaría el problema de búsqueda en un escenario más complejo. También para versiones posteriores Will-E podría tener integrado otros tipos de recomendadores (no solo receta), ya sea de cine, televisión, música, o incluso sabiendo un poco más del cliente (usuario final) podría saber a qué hora es mejor limpiar o realizar determinada acción en concreto, esto también fue difícil de hacer por la naturaleza casi-aleatoria del humano, o sea, actúa con cierto sentido, gracias a que lo hace para satisfacer sus necesidades, pero esto lo puede hacer en cualquier orden.

Otro aspecto mejorable es el tiempo que duró la simulación, en este caso se hacía bastante difícil simular más de lo que se hizo, puesto que la API de Gemini tiene una cota de solicitudes diarias la cual al consumirla no permite seguir haciendo solicitudes, por lo que la simulación no podía durar más.

## 5.0 Bibliografía

1. Google Developers: *Get Started with the Gemini API: Python*

2. Kwang H. Lee (Berlín, Alemania, 2005) : *First Course on Fuzzy Theory and Applications*

3. Luciano García Garrido, Luis Martí Orosa, Luis Pérez Sánchez (La Habana, Cuba): *Temas de Simulación*

4. Rafael H. Bordini, Jomi Fred Hübner, Michael Wooldridge (Londres, Inglaterra, 2007): *Programming multi-agent systems in AgentSpeak using Jason*

5. Stuart Russell, Peter Norving (2020): *Artificial Intelligence: A Modern Approach*