2do Proyecto de Simulación:

Kindergarden

Miguel Tenorio Potrony

Facultad de Matemática y Computación
Universidad de La Habana

Curso 2019/2020

Índice

1. Generales del Estudiante **1**

2. Orden del Problema Asignado **1**

3. Principales Ideas seguidas para la solución del problema **1**

4. Modelos de Agentes considerados **1**
4.1. Modelo Reactivo 1
4.2. Modelo de Razonamiento Práctico 2

5. Ideas seguidas para la implementación **4**
5.1. Ambiente 4
5.2. Agentes 6

6. Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema **10**
6.1. Simulaciones 11

7. Enlace al repositorio del proyecto en Github **15**

1. Generales del Estudiante

- **Nombre y apellidos**: Miguel Tenorio Potrony

- **Grupo**: 412

- **Email**: m.tenorio@estudiantes.matcom.uh.cu

2. Orden del Problema Asignado

- **Agentes**

3. Principales Ideas seguidas para la solución del problema

Dado que el ambiente definido en la orientación puede ser en cierta medida vólatil debido a los cambios de ambiente aleatorios, se decidió que los agentes más indicados para completar la tarea en estas condiciones son aquellos que sean capaces de reorganizarse cuando sus intenciones caduquen, y tomar ventajas de las situaciones casuales y las nuevas oportunidades, de modo que puedan enfrentar de la mejor forma posible estos cambios. Por tanto se diseñaron dos tipos de agentes, uno más capaz y eficiente que el otro, inspirados en la idea anterior.

Las reglas de la simulación están bien definidas en la orientación del proyecto, por lo que el principal problema a resolver es el diseño e implementación de un agente (el Robot de Casa) que sea capaz de completar su tarea en un ambiente que cumpla las condiciones orientadas. Por tanto, sobre el ambiente deben interesar los detalles de su implementación, tema trabajado en la sección 5. El grueso de este informe consiste en explicar la arquitectura y el programa de los dos agentes realizados, los enfoques que toman para completar su tarea y los resultados de las simulaciones.

4. Modelos de Agentes considerados

Se consideraron e implementaron dos modelos de agentes:

- Modelo Reactivo [1]
- Modelo de Razonamiento Práctico [2]

4.1. Modelo Reactivo

El diseño de este modelo está inspirado en la Arquitectura de Brook, por tanto se implementó una jerarquía por categorías, con los comportamientos ordenados por capas. En este caso, las capas están ordenadas según la prioridad, siendo las primeras capas las que representan los comportamientos priorizados. Se procede a definir estas capas. Se refiere a *la casilla actual* como la casilla donde está ubicado el robot en el turno del presente.

1. Si en el turno anterior el robot decidió limpiar la casilla actual, entonces esta se limpia.

2. Si hay suciedad en la casilla actual y no está cargando a un niño, entonces el robot decide limpiarla en el próximo turno.

3. Si en la casilla actual hay un niño, no es una casilla del corral, y el robot no esta cargando a un niño, entonces automáticamente carga a ese niño.

4. Si la casilla actual es una casilla del corral, no tiene un niño dentro, y el robot está cargando a un niño, entonces decide dejarlo en esa casilla del corral.

5. Si no tiene casillas adyacentes a la cual moverse (e.g rodeado de obstáculos) entonces el robot decide esperar a que el ambiente cambie lo suficiente para poder moverse.

6. Si el robot está cargando un niño, entonces se mueve hacia la casilla del corral vacía más cercana. Aprovecha la propiedad de información completa del ambiente para construir el camino más corto y válido en el momento actual hacia ese objetivo. Lo mismo se aplica en lo adelante al decir que el robot se mueve hacia un objetivo en particular.

7. Si hay niños fuera del corral, entonces el robot se mueve hacia las casilla más cercana de estos.

8. Si hay suciedad, entonces el robot se mueve hacia la suciedad más cercana.

9. Si el robot no puede responder a ningún comportamiento representado por las capas anteriores, entonces es debido a que completó su trabajo satisfactoriamente, por tanto espera a que se termine la simulación.

Del análisis de las capas anteriormente mencionadas, se desprende que este robot prioriza la recogida de niños ante la limpieza de suciedad. Se escogió esta preferencia debido a que los niños pueden generar suciedad en cada turno, por tanto es preferible que el robot elimine potenciales generadores de suciedad, y así evitar la acumulación de esta en el ambiente. Además, se aprovecha la ventaja que representa poder moverse a través de dos casillas consecutivas cuando se está cargando un niño. Si, en su camino hacia un niño, encuentra suciedad la recogerá. Inevitablemente habría que poner una prioridad sobre la otra, así que se decidió el orden enunciado debido a los argumentos previos.

De acuerdo al criterio de flexibilidad de un agente inteligente por [4], el agente implementado siguiendo este modelo solo cumple con ser **Reactivo**, i.e capaz de percibir su ambiente y responder de un modo oportuno a los cambios que ocurren para lograr sus objetivos.

Según [5], el programa de este agente es del tipo **Single reflex agent**. En el siguiente epígrafe (5) se podrán ver los detalles de implementación.

4.2. Modelo de Razonamiento Práctico

La arquitectura de este modelo es la BDI (creencia-deseo-intención, por sus siglas en inglés) [2]. Debido a que el ambiente estudiado puede cambiar en su totalidad de forma aleatoria, periódica, y posiblemente frecuente, este modelo persigue construir un tipo de agente *precavido*, según el criterio de David Kinny y Michael Georgeff [3]. Inspirándose en las ideas planteadas, el agente en este diseño porta dos intenciones principales estrechamente relacionadas a una percepción: el porcentaje de suciedad en el ambiente:

- Si este valor es mayor o igual que 45, el objetivo principal del agente pasa a ser la recogida de suciedad para disminuir este índice negativo, que lo puede llevar a ser despedido si no se controla a tiempo. En este caso el agente adopta un modo *precavido*, en el cual hace énfasis en decidir cuál es la mejor acción en ese momento para ser fiel a la intención antes mencionada, y a la vez seguir cumpliendo la tarea principal en el proceso (recoger a todos los niños y limpiar toda la casa). El agente está en peligro y debe usar todos sus recursos para salir del aprieto.

- En caso contrario, el agente adopta un modo *atrevido*. No se siente en aprietos y puede enfocarse en lo que considera el mejor objetivo a largo plazo: recoger a los niños. Estos son el elemento del ambiente que más provocan cambios negativos para el progreso de la tarea del robot. No se detiene nunca a recoger suciedad como su similar Reactivo. Sin niños fuera de control, ya se podrá encontrar cuál es la mejor manera de limpiar la casa sin que los planes sean afectados por la acción de los niños.

A continuación se detallarán las componentes de este modelo, adaptadas, por supuesto, a resolver el problema dado al agente.

- **Revisión de creencias**
  En este componente el agente actualiza sus estados internos y construye información a partir de la cual el agente podrá generar un número de opciones o planes. En este punto sería útil conocer cuáles son estos estados.

  - `t` → El robot sabe cómo aprender o detectar cuál es el valor de t en el ambiente
  - `babies` → Número de niños fuera del corral
  - `mess` → Porcentaje de suciedad en el ambiente
  - `dirt_groups` → Grupos de suciedad formados a partir de un criterio de agrupación
  - `baby_groups` → Grupos de niños formados a partir de un criterio de agrupación
  - `paths_to_babies` → Caminos hacia los niños
  - `dirt_in_path_to_babies` → Cantidad de suciedad en los caminos hacia los niños
  - `paths_to_dirt` → Caminos hacia los elementos de suciedad

- **Generación de opciones**
  En esta fase el agente construye a partir de sus creencias los mejores planes que puede formar dada la información que posee: el mejor[1] camino hacia la suciedad, el mejor camino hacia un niño, y el mejor camino, que contenga suciedad, hacia un niño.

- **Filtrado de opciones**
  En esta etapa el agente determina cuál es su intención a partir de las creencias, y escoge qué acción va a ejecutar en base a la intención seleccionada y los planes formados en la componente anterior.

- **Función de selección**
  En esta fase se ejecuta la acción decidida por el agente en todo el proceso de su deliberación y razonamiento.

[1] Este criterio se explica en 5

De acuerdo al criterio de flexibilidad de un agente inteligente por [4], el agente implementado siguiendo este modelo cumple con ser **Reactivo** y **Pro-Activo**, i.e capaz de percibir su ambiente y responder de un modo oportuno a los cambios que ocurren para lograr sus objetivos además de mostrar un comportamiento *Goal-Directed* tomando la iniciativa para lograr sus objetivos.

Según [5], el programa de este agente es del tipo **Goal-based agent**. En el siguiente epígrafe (5) se podrán ver los detalles de implementación.

5. Ideas seguidas para la implementación

5.1. Ambiente

Para la representación del ambiente, se creó la clase `Env` del módulo `env`, cuya propiedad principal `house` es una matriz de $N \times M$ cuyos valores son objetos `Cell`. Un `Cell` contiene 3 atributos:

- `value` → Es una combinación de los siguientes valores: ('B': niño o bebé, '*': suciedad, 'C': casilla del corral, 'R': robot, '#': obstáculo, '-': casilla vacía). Representa el valor de una casilla. Se dice que es una combinación ya que si en la misma casilla conviven dos o más elementos, entonces este toma como valor la cadena formada por cada uno de estos elementos unidos por el símbolo '-', a excepción del caso en que un elemento o varios ocupe el lugar de una casilla vacía, donde su valor queda omitido. Ejemplos de la forma (situación-valor de la casilla):
  - El robot se mueve hacia una casilla vacía: 'R'
  - El robot se mueve hacia una casilla donde hay un niño: 'B-R'(Cuando el símbolo del robot es el último en la cadena se debe a que solo está conviviendo con el resto de los elementos de la casilla sin cometer ninguna acción en específico).
  - El robot carga a un niño: 'R-B'
  - El robot deja a un niño en una casilla del corral: 'C-B-R'
- `dirty` → `True` si la casilla posee suciedad, `False` en caso contrario.
- `isFixed` → `True` si la casilla es una casilla del corral, `False` en caso contrario.

`Env` posee el método `simulate` (figura 1) que se encarga de manejar el flujo de la simulación. Este flujo es el siguiente: Sea t_i el turno de la simulación correspondiente a la unidad de tiempo $i \in [1, t * 100]$ donde t es el parámetro del ambiente responsable de dictar cuando ocurre el cambio aleatorio del ambiente. Al comienzo del turno se mueve el robot mediante su método `action`, y luego los niños mediante el método `natural_change` (figura 2), que se encarga de mover a los niños, generar suciedad y actualizar las condiciones de la simulación, la cuál corre mientras el robot no ha sido despedido, no ha completado todo su trabajo, o el tiempo límite no ha sido alcanzado. Llegado el momento, luego del cambio natural del ambiente, ocurre el cambio aleatorio mediante el método `random_change`.

Se puede observar en la figura 2 el flujo del cambio natural del ambiente, sin embargo no desprende detalles de la implementación concreta de esta funcionalidad, por lo tanto se ofrecerá a continuación una explicación sobre estos.

La función `move_babies` se encarga del movimiento de los niños libres (fuera del corral). La posibilidad del movimiento de un niño se determina a partir de

```
def simulate(self, interactive):
    while self.running and self.time < self.t * 100:
        self.time += 1
        self.robot.action(self.house)
        self.natural_change()

        if self.time % self.t == 0:
            self.random_change()
```

Figura 1: Código fuente esencial de `simulate`.

```
def natural_change(self):
    n, m = get_length(self.house)
    babies = [(i, j) for i in range(n) for j in range(m)
     if self.house[i][j].value == BABY]

    move_babies(self.house, babies, log, self.p)
    free_babies = get_free_babies(self.house, babies)
    babies_in_order = len(free_babies) == 0
    mess_up(self.house, free_babies)

    total_mess = count_dirt(self.house)
    if total_mess == 0 and babies_in_order:
        #The robot completed its job successfully!!!
        self.running = False
        self.succeded = True
        return

    n, m = get_length(self.house)
    if total_mess * 100 // (n * m) >= 60:
        #The robot is fired!!!
        self.running = False
```

Figura 2: Código fuente esencial de `natural_change`.

una variable aleatoria Bernoulli, cuyo párametro p tiene como valor por defecto 0,5 y puede ser alterado por los argumentos dados al programa al inicio de la ejecución del mismo. En caso de que el niño se vaya a mover, se escoge una casilla adyacente, vacía o con un obstáculo, y se efectúa el movimiento. Si la casilla objetivo es un obstáculo, se analiza si este puede ser desplazado. Si esta acción fracasa, el niño se mantiene en su lugar y no ocurre movimiento.

La función `get_free_babies` forma grupos de niños donde dos niños están en el mismo grupo si la distancia euclidiana de sus posiciones es menor o igual que 2 (están en la misma cuadrícula de 3×3). Un pequeño detalle es que se usan las posiciones de los niños antes de los efectos de `move_babies`, ya que estos grupos están destinados para servir de apoyo al proceso de generar suciedad, y se orientó que cuando un niño se mueve, una casilla de la cuadrícula anterior de 3×3 puede haber sido ensuciada. El código se puede observar en la figura 3.

La función `mess_up` simplemente recorre cada grupo de niños formado por `get_free_babies` y trata de generar suciedad en la cuadrícula representada por cada grupo. Por cada grupo, determina cuanta suciedad se puede crear, y escoge de forma aleatoria una casilla adyacente a un niño (o posición anterior de un

```python
def get_free_babies(house, babies):
    free_babies = {b for b in babies if not house[b[0]][b[1]].isFixed}
    groups = set()
    for baby in free_babies:
        g = [baby]
        for other in free_babies - {baby}:
            if distance(baby, other) <= 2:
                g.append(other)
        groups.add(frozenset(g))
    return groups
```

Figura 3: Código fuente esencial de `simulate`.

niño luego de haberse movido) dentro de esa cuadrícula y si esta está vacía, se genera suciedad en ella, si no, se descarta esa posible generación de suciedad.

El cambio aleatorio dado por `random_change` se efectúa primero determinando una posición (x, y) aleatoria para cada casilla c, y luego se intercambian las posiciones de la casilla c con la casilla ubicada en la coordenada (x, y) de la casa. Luego se ubica la posición de cada casilla del corral, se escoge una, y por cada casilla del corral no accesible a esta mediante casillas consecutivas o adyacentes, se intercambia su posición con la de una casilla que no es del corral, accesible mediante casillas del corral a esta primera casilla del corral. De este modo se mantiene el corral formado por casillas consecutivas luego de la variación aleatoria.

5.2. Agentes

Un Robot de Casa es una implementación de la clase abstracta `Robot`, cuyo único método es `action` el cual recive como percepción la casa como tal en el turno actual. Esta es la percepción ya que el ambiente es de información completa. Este método representa la reacción de un agente, el cual toma como entrada un conjunto de percepciones y da como salida una acción sobre el ambiente o sobre sí mismo.

El agente `Reagent` es el que sigue el modelo reactivo y hereda de la clase `Robot`. Este, además de implementar el método `action`, contiene los métodos `move` y `try_move`, los cuales se encargan de recrear y controlar el movimiento del robot sobre el ambiente.

El modelo **Reactivo** se implementó de una manera muy directa. Entre los pocos detalles significativos está que el robot determina cuál es la suciedad, o el niño más cercano mediante el algoritmo **BFS**. Los agentes diseñados para esta simulación, mantienen su posición actual, y un atributo `will_clean` cuyo valor es `True` si el robot decidió limpiar la suciedad de la casilla actual en el próximo turno.

El agente `Practical` sigue el modelo de **Razonamiento Práctico** y hereda de la clase `Reagent` el diseño de un `Robot` y el manejo de los movimientos del robot. Implementa su propio método `action`, además de otros, siguiendo la arquitectura **BDI** antes enunciada.

A continuación se explican los detalles significativos de implementación de las componentes del modelo de **Razonamiento Práctico**.

- **Revisión de creencias**
  El robot aprende cuál es el valor de t en el ambiente manteniendo las coordenadas de su posición al finalizar el turno anterior, las coordenadas de su posición actual y su tiempo de vida. Si su posición actual al empezar el turno, no es la misma que la anterior, entonces es debido a que ocurrió un cambio de ambiente aleatorio. En el caso remoto de que este cambio haya ocurrido y el robot mantiene la misma posición, lo que sucede que el robot se demora más en detectar cuál es el valor de esa propiedad del ambiente, ya que este proceso de aprendizaje es constante.

  Los atributos `dirt_groups` y `baby_groups` se construyen mediante la función `group_target` que dado una casa, un elemento del ambiente, y un criterio de agrupación γ, forma grupos de estos elementos, donde dos elementos están en el mismo grupo si la distancia de *Manhattan*[2] de sus posiciones es menor o igual que γ. (Nótese su similitud con 3)

```
def group_target(house, target, threshold):
    n, m = get_length(house)
    matches = {(i, j) for i in range(n) for j in range(m)
        if house[i][j].value == target}
    groups = set()
    for m in matches:
        g = [m]
        for other in matches - {m}:
            if distance(m, other) <= threshold:
                g.append(other)
        groups.add(frozenset(g))
    return groups
```

Figura 4: Código fuente esencial de `group_target`.

El valor γ para la construcción de `dirt_groups` es $t/2$ o 10 si t no ha sido detectado todavía. De este modo, se agrupa la suciedad de forma tal que en t turnos se puedan limpiar $t/2$ elementos de suciedad. Esto permite al agente formar un plan de recogida de suciedad que sea inmune a los cambios aleatorios del ambiente. Aplicando esta estrategia, se construyen las creencias del agente, ya que no tiene sentido trazar un plan o una secuencias de acciones intencionales sobre información que deja de ser válida cada t turnos.

El valor γ para `baby_groups` es 2, con el objetivo de agrupar niños cercanos en una cuadrícula de 3 X 3, ayudando de este modo a escoger un camino hacia un niño con el objetivo de recogerlo y evitar la generación de suciedad adicional dada por la concentración de niños en una de esas cuadrículas.

Para la construcción de los caminos válidos (caminos que pueden ser recorridos por el robot dada la circunstancia actual) más cercanos tanto a los niños como a la suciedad, se utiliza el algoritmo **DFS**. Se trata de obtener todos los caminos simples a estos objetivos, para poder decidir entre 2 caminos de igual longitud cuál es el más adecuado dada la intención del

[2]Se escogió esta función de distancia dado que el robot no puede moverse hacia casillas diagonales a su posición, y la información de estos grupos está destinada a la determinación de caminos

agente y sus creencias. Por un tema de eficiencia, solo se buscan caminos de longitud cercanas a la longitud del camino más corto, ya que al agente solo le interesan los caminos que constituyan las opciones más eficientes y viables disponibles.

- **Generación de opciones**
  Como fue explicado en la sección anterior, el agente construye 3 planes efectivos. Se procede entonces a entrar en los detalles de estos planes.

  El mejor camino hacia la suciedad es el de menor longitud a un elemento de suciedad. De existir varios de estos (de la misma longitud), se escoge el camino hacia el elemento que mayor concentracción de suciedad tenga alrededor -según el criterio γ-. En la figura 5 se puede apreciar el código de este algoritmo.

```
#Get best path to dirt
considering = list()
for p in options['paths_to_dirt']:
    for g in options['dirt_groups']:
        last = list(p)[-1]
        if last in g:
            #(length of path, concentration of dirt, path)
            considering.append((len(p), -len(g), p))
            break
considering.sort()
options['best_path_dirt'] = None if considering == [] else considering[0][2]
```

Figura 5: Mejor camino hacia la suciedad.

  El mejor camino, que contenga suciedad, hacia un niño es el de menor longitud al niño. De existir varios de estos (de la misma longitud), se escoge el camino hacia el niño con más presencia de suciedad. En la figura 6 se puede apreciar el código de este algoritmo.

```
#Get best path to baby sorted by occurrence of dirt in the path
considering.clear()
for p, d in zip(options['paths_to_babies'], options['dirt_in_path_to_babies']):
    #(length of path, number of dirt in path, path)
    considering.append((len(p), -d, p))
considering.sort()
options['best_dirty_path_babies'] = None if considering == []
 else considering[0][2]
```

Figura 6: Mejor camino que contenga suciedad, hacia un niño.

  El mejor camino hacia un niño es el de menor longitud al niño. De existir varios de estos (de la misma longitud), se escoge el camino hacia el niño que mayor concentracción de niños tenga alrededor -según el criterio γ-. En la figura 7 se puede apreciar el código de este algoritmo.

- **Filtrado de opciones**
  Las posibles acciones que el agente puede ejecutar son:

```
#Get best path to baby
considering.clear()
for p in options['paths_to_babies']:
    for g in options['baby_groups']:
        last = list(p)[-1]
        if last in g:
            #(length of path, concentration of babies, path)
            considering.append((len(p), -len(g), p))
            break
considering.sort()
options['best_path_babies'] = None if considering == [] else considering[0][2]
```

Figura 7: Mejor camino hacia un niño.

- `DROP` Dejar al niño cargado en la casilla actual
- `GOTO_CORRAL` Dirigirse al corral vacío más cercano
- `CLEAN` Limpiar la suciedad en la casilla actual
- `MOVE` Moverse siguiendo el camino planeado
- `BLOCK` Esperar en el lugar, no se puede realizar ninguna acción que cumpla con la intención actual.

En esta etapa se escoge una de esas acciones a ser ejecutada.

Las tres primeras acciones listadas se escogen automáticamente si se alcanzan las condiciones que permitan realizarlas.

`DROP` El robot decide esta acción si está cargando un niño y en el turno anterior se activó la propiedad `drop`. Esta propiedad toma el valor `True` cuando el robot está cargando un niño y no existe un camino válido hacia los corrales. La estrategia en este caso es ir hacia una casilla vacía, soltar al niño allí, y activar el modo `CAUTIOUS` (*precavido*) para concentrarse en recoger basura hasta que ocurra un cambio aleatorio que permita llegar a los corrales.

`GOTO_CORRAL` Se decide tomar esta acción si se está cargando un niño y la propiedad `drop` tiene valor `False`.

`CLEAN` Si el atributo `will_clean` tiene valor `True`, entonces en el turno anterior se llegó a una casilla con suciedad y se decidió limpiarla en el presente turno. Por tanto el agente decide que va a ejecutar esta acción.

Si no existen las condiciones que permiten escoger las acciones explicadas en los tres últimos párrafos, entonces es momento para que el agente escoja cuál es el mejor plan a seguir. Al comienzo de la sección 4.2 se explicó en qué consisten las dos principales intenciones del agente reflejadas en los modos *precavido* y *atrevido*. Las creencias actualizadas en las etapas anteriores permiten escoger cuál será la intención del agente en este turno.

En caso de que *precavido* sea el modo escogido, entonces se decide qué camino a tomar es mejor entre el mejor camino hacia la suciedad y el mejor camino, que contenga suciedad, hacia un niño. Es válido recordar

que estas opciones son tuplas del tipo `(longitud del camino, métrica especial, camino)`. Se insertan estas tuplas (de existir ambas) en una lista, se ordena y se escoge la menor tupla como el mejor plan disponible.

Si se escoge el modo *atrevido*, simplemente el mejor plan es ir hacia el mejor camino hacia un niño.

Si en este proceso de decisión, para ambos casos, no hay un plan a escoger, entonces el robot decide tomar la acción `BLOCK`.

- **Función de selección**
  En esta fase se ejecuta la acción decidida por el agente en el proceso anterior.

6. Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema

Se crearon 15 ambientes iniciales para realizar las simulaciones y medir de ese modo el desempeño de los agentes. Se realizaron por cada ambiente inicial, 30 simulaciones. Una simulación realizada consiste en crear un ambiente dado una especificación de ambiente inicial, luego dada la casa inicial, se simula el proceso con un tipo de agente, y luego con el otro con la misma casa inicial. De este modo se mide el comportamiento de los dos agentes con las mismas condiciones iniciales. Es lo más cercano que se puede tener a ambientes similares para los agentes, ya que muchos de los factores de cambio ajenos a los agentes se realizan de forma aleatoria. Se procede entonces a listar los resultados para cada ambiente inicial.

Los ambientes se diseñaron de forma tal que pudieran representar un desafío para los agentes y todos parten de las misma especificación inicial. Lo que varía son ciertos párametros. Como se podrá observar se formularon cuatro dificultades: fácil, normal, difícil, y muy desafiantes.

Es válido aclarar que en la estadítica del número de veces que el robot fue despedido, se encuentra también el número de veces que se le acabó el tiempo al robot para completar la tarea. Una tarea completada significa que el robot limpió toda la casa y ubicó a todos los niños en el corral.

La especificación común es la siguiente:

- $N = 9$
- $M = 8$
- Porcentaje de casillas que aparecen sucias (suciedad): 30
- Porcentaje de obstáculos (obstáculos): 20
- Número de niños: 5
- $t = 5$
- Párametro p de la v.a Bernoulli encargada de decidir la posibilidad de movimiento para un niño: 0,5

6.1. Simulaciones

- Ambientes fáciles:

  1. Ambiente por defecto (sin variación de los párametros comunes):
     - Agente Reactivo:
       - Media del porciento de casillas sucias: 6,2
       - Número de veces despedido: 3
       - Número de tareas completadas: 27
     - Agente de Razonamiento Práctico:
       - Media del porciento de casillas sucias: 0,1
       - Número de veces despedido: 1
       - Número de tareas completadas: 29
  2. $t = 6$, suciedad al 45 %:
     - Agente Reactivo:
       - Media del porciento de casillas sucias: 10,2
       - Número de veces despedido: 5
       - Número de tareas completadas: 25
     - Agente de Razonamiento Práctico:
       - Media del porciento de casillas sucias: 2,1
       - Número de veces despedido: 1
       - Número de tareas completadas: 29
  3. $t = 6$, obstáculos al 30 %:
     - Agente Reactivo:
       - Media del porciento de casillas sucias: 0,0
       - Número de veces despedido: 0
       - Número de tareas completadas: 30
     - Agente de Razonamiento Práctico:
       - Media del porciento de casillas sucias: 0,0
       - Número de veces despedido: 0
       - Número de tareas completadas: 30
  4. $t = 6$, suciedad al 15 %, 8 niños:
     - Agente Reactivo:
       - Media del porciento de casillas sucias: 6,1
       - Número de veces despedido: 3
       - Número de tareas completadas: 27
     - Agente de Razonamiento Práctico:
       - Media del porciento de casillas sucias: 0,0
       - Número de veces despedido: 1
       - Número de tareas completadas: 29

- Ambientes normales:

  1. $t = 4$:
     - Agente Reactivo:
       - Media del porciento de casillas sucias: 18,4
       - Número de veces despedido: 9
       - Número de tareas completadas: 21

- Agente de Razonamiento Práctico:
  - Media del porciento de casillas sucias: 0,0
  - Número de veces despedido: 1
  - Número de tareas completadas: 29

2. $t = 4$, $p = 0{,}75$:
   - Agente Reactivo:
     - Media del porciento de casillas sucias: 6,1
     - Número de veces despedido: 3
     - Número de tareas completadas: 27
   - Agente de Razonamiento Práctico:
     - Media del porciento de casillas sucias: 0,0
     - Número de veces despedido: 0
     - Número de tareas completadas: 30
3. $t = 4$, $p = 0{,}75$, suciedad al 45 %:
   - Agente Reactivo:
     - Media del porciento de casillas sucias: 17,0
     - Número de veces despedido: 9
     - Número de tareas completadas: 21
   - Agente de Razonamiento Práctico:
     - Media del porciento de casillas sucias: 2,2
     - Número de veces despedido: 3
     - Número de tareas completadas: 27
4. $t = 4$, $p = 1$, suciedad al 10 %, obstáculos al 10 %:
   - Agente Reactivo:
     - Media del porciento de casillas sucias: 20,4
     - Número de veces despedido: 10
     - Número de tareas completadas: 20
   - Agente de Razonamiento Práctico:
     - Media del porciento de casillas sucias: 10,3
     - Número de veces despedido: 6
     - Número de tareas completadas: 24

- Ambientes difícles:

1. $t = 3$:
   - Agente Reactivo:
     - Media del porciento de casillas sucias: 25,5
     - Número de veces despedido: 15
     - Número de tareas completadas: 15
   - Agente de Razonamiento Práctico:
     - Media del porciento de casillas sucias: 13,1
     - Número de veces despedido: 10
     - Número de tareas completadas: 20
2. $t = 3$, $p = 1$, suciedad al 10 %, obstáculos al 10 %:
   - Agente Reactivo:
     - Media del porciento de casillas sucias: 33,2
     - Número de veces despedido: 17

        - Número de tareas completadas: 13
    - Agente de Razonamiento Práctico:
        - Media del porciento de casillas sucias: 13,1
        - Número de veces despedido: 10
        - Número de tareas completadas: 20
3. $t = 2$:
    - Agente Reactivo:
        - Media del porciento de casillas sucias: 33,8
        - Número de veces despedido: 30
        - Número de tareas completadas: 0
    - Agente de Razonamiento Práctico:
        - Media del porciento de casillas sucias: 14,8
        - Número de veces despedido: 29
        - Número de tareas completadas: 1
4. $t = 2$, $p = 1$:
    - Agente Reactivo:
        - Media del porciento de casillas sucias: 27,2
        - Número de veces despedido: 29
        - Número de tareas completadas: 1
    - Agente de Razonamiento Práctico:
        - Media del porciento de casillas sucias: 19,9
        - Número de veces despedido: 28
        - Número de tareas completadas: 2
5. $t = 2$, suciedad al 40 %, 6 niños:
    - Agente Reactivo:
        - Media del porciento de casillas sucias: 58,3
        - Número de veces despedido: 30
        - Número de tareas completadas: 0
    - Agente de Razonamiento Práctico:
        - Media del porciento de casillas sucias: 40,5
        - Número de veces despedido: 29
        - Número de tareas completadas: 1

- Ambientes muy desafiantes:

1. $t = 2$, suciedad al 10 %, 8 niños:
    - Agente Reactivo:
        - Media del porciento de casillas sucias: 54,0
        - Número de veces despedido: 30
        - Número de tareas completadas: 0
    - Agente de Razonamiento Práctico:
        - Media del porciento de casillas sucias: 40,1
        - Número de veces despedido: 30
        - Número de tareas completadas: 0
2. $t = 1$, $p = 1$, suciedad al 40 %, 6 niños:
    - Agente Reactivo:
        - Media del porciento de casillas sucias: 54,8

    - Número de veces despedido: 30
    - Número de tareas completadas: 0
- Agente de Razonamiento Práctico:
    - Media del porciento de casillas sucias: 50,2
    - Número de veces despedido: 30
    - Número de tareas completadas: 0

De estos datos se puede observar que el agente de **Razonamiento Práctico** es más eficiente y exitoso que su homólogo **Reactivo**. Algunos parámetros provocan diferentes resultados en ambos agentes, pero definitivamente el factor que más afecta el desempeño de los agentes es el valor de t. Mientras más vólatil es el ambiente, menos rinde el robot. Este resultado es algo obvio. Si bien para valores altos de t un cambio aleatorio del ambiente podría beneficiar al robot dado que tendría más tiempo para completar su función, o podría alterar un estado desfavorable del ambiente para él; con valores muy bajos (menores que dos) tiene menos tiempo y casi no puede seguir un plan de acción. El tamaño de la casa se mantuvo constante para medir comportamientos ajenos a este, sobre todo en un entorno donde se trabaja con proporciones. Los hechos de que el robot se mueva en un paso o dos solamente, que al haber más casillas vacías se genere más rápido suciedad, y que mientras más grande sea la casa mayor procesamiento es requerido, son algunos de los factores que también determinaron esta decisión. Para contrarrestar los factores anteriores habría que aumentar lo suficientemente el valor de t como para mantener cierta proporción tamaño - t, y esto implica muchos más turnos y mayor demora de la simulación. Se escogió el tamaño que se consideró más viable para la suite de simulación.

El hecho de que los niños se muevan con mayor probabilidad, no pareció afectar a ningún agente.

Niveles altos de suciedad inicial tienen mucho impacto en el rendimiento del agente **Reactivo**. Su desempeño es menor también en ambientes con condiciones iniciales propicias para la generación de basura (poca suciedad inicial, más niños, pocos obstáculos).

Entre los factores específicos que provocan un menor rendimiento del agente de **Razonamiento Práctico** está la existencia de un ambiente propicio para un nivel alto de movimiento de niños: pocos obstáculos, poca suciedad inicial y $p = 1$. El otro factor es el límite de turnos. Esto se nota en el promedio del porcentaje de suciedad en los ambientes difíciles, donde toma valores cercanos a 30 (el valor medio entre los valores extremos de esta métrica: 0 y 60), lo que implica que buena parte de las simulaciones fallidas se deben a que el robot se quedó sin tiempo para completar la tarea.

Se puede concluir afirmando que, aún en ambientes que cambian con mucha frecuencia, un agente pro-activo y reactivo es una buena opción a valorar ante un clásico agente reactivo. Se podría pensar lo contrario, ya que un agente que no forma planes o toma decisiones basadas en intenciones, y que solo responda ante simples reflejos o percepciones, tiene los ingredientes necesarios para triunfar en un entorno donde los planes se hacen añicos constantemente. No obstante, los resultados dicen lo contrario. A veces no es suficiente con responder de modo oportuno a los cambios, si no además mostrar un comportamiendo *Goal-Directed* tomando la iniciativa para lograr los objetivos.

7. Enlace al repositorio del proyecto en Github

https://github.com/stdevAntiD2ta/Kindergarden

Referencias

[1] Luciano García Garrido, Luis Martí Orosa, Luis Pérez Sánchez. Temas de Simulación, capítulo 6, epígrafe 6.5, página 73.

[2] Luciano García Garrido, Luis Martí Orosa, Luis Pérez Sánchez. Temas de Simulación, capítulo 6, epígrafe 6.4, página 67.

[3] Luciano García Garrido, Luis Martí Orosa, Luis Pérez Sánchez. Temas de Simulación, capítulo 6, epígrafe 6.4.2, página 69.

[4] Luciano García Garrido, Luis Martí Orosa, Luis Pérez Sánchez. Temas de Simulación, capítulo 6, epígrafe 6.1.2, página 49.

[5] Stuart J. Russell, Peter Norvig. Artificial Intelligence A Modern Approach, 3rd Edition-Pearson Education (2016), capítulo 2.