Planificación en Entornos con Incertidumbre

(IA) Inteligencia Artificial
Universidad de Sevilla
Sevilla, España

Candelario Luna, Luis
luicanlun@alum.us.es

Carrasco Márquez, Antonio
antcarmar2@alum.us.es

28 de Junio 2019

Resumen **- Este documento recopila el proceso, implementación y experiencias de los autores del mismo tras realizar el trabajo propuesto llamado "Planificación en Entornos con Incertidumbre" por el profesor Ignacio Pérez Hurtado de Mendoza para la asignatura Inteligencia Artificial.**

1 Introducción

Nuestro objetivo consiste en realizar una implementación en Python que permita resolver y mostrar los resultados de un problema planteado como POMDP mediante uno de los dos algoritmos, bien sea POMCP (Partially Observable Monte-Carlo Planning) o PBVI (Point-Based Value Iteration) y mostrar sus resultados de dos formas distintas: (1) Simulación interactiva, (2) Simulación silenciosa o benchmark. Para ello, hemos utilizado y modificado una librería que nos permitirá ejecutar simulaciones dado un problema en formato "POMDP" llamada "PyPOMDP" [1] con el fin de obtener y mostrar los resultados de ambas formas.

2 Preliminares

2.1 Compresión del enunciado y análisis

Antes de realizar la implementación, nos reunimos para analizar el enunciado del trabajo y poder comprender aquello que nos pedían además de los requisitos de la entrega y los criterios de evaluación. Para ello, primero hicimos un análisis sobre los Procesos de Decisión de Markov para posteriormente enfocarnos en los Procesos de Decisión de Markov Parcialmente Observables (POMDP).

Tras analizarlo, llegamos a la conclusión de que, en resumen, los POMDP no se conoce el estado actual. Para ello planificador debe representar su creencia o *belief* a través de una distribución de probabilidad sobre el conjunto de estados. Cada vez que realizamos una acción, se desconoce el estado que se alcanza, pero se obtiene una observación que permite actualizar el *belief*.

2.2 Instalación del entorno de desarrollo

Antes de realizar la implementación, tuvimos que descargar e instalar el software en nuestros respectivos equipos para poder llevarla a cabo. Para ello, decidimos utilizar el entorno llamado "PyCharm" [2] y debido a que íbamos a trabajar sobre sistemas operativos Windows y habíamos trabajado antes con entornos similares era nuestra mejor opción.

2.3 Importación de la librería

Una vez instalado el entorno de desarrollo, accedimos y analizamos la librería "PyPOMDP" [1] con el objetivo de conocer si realmente nos era útil para realizar la implementación. Para ello, realizamos un *fork* del repositorio [3] con el fin de poder trabajar directamente con él. Tras esto, decidimos importar la librería en nuestro entorno de desarrollo y comenzar a implementar no sin antes, consultar el fichero "Readme.md" para instalar los paquetes de Python necesarios, así como probar el ejemplo de instrucción para un problema ya implementado en la librería con cada uno de los algoritmos deseados para ver si realmente funcionaba. En este caso, fue el problema del Tigre de 3 puertas "Tiger-3D.POMP", la prueba fue exitosa y comenzó a mostrar los resultados en la consola ejecutando previamente las siguientes instrucciones:

>python main.py pomcp –env Tiger-3D.POMDP
>python main.py pbvi –env Tiger-3D.POMDP

3 Metodología

Una vez probado el problema del tigre de 3 puertas, nos percatamos de que también estaba implementado el problema del tigre de 2 puertas, sin embargo, este no funcionaba. Para ello, tomamos como referencia el problema del tigre con 3 puertas y corregimos el formato del documento "Tiger-2D.POMP". Posteriormente ejecutamos la instrucción principal de nuevo mostrando los resultados en la consola. En estos resultados apreciamos que la salida que aparece a continuación se mostraba un número determinado de veces sin parar variando entre los estados y las acciones disponibles, dado por el valor por defecto (100) del parámetro "max_play" de la clase "main.py".

```
# Step = 4025
Taking action: listen
Observation: tiger-left
Reward: -1.0
New Belief: [0.956679, 0.043321]
```

Figure 1: Fragmento de la salida inicial

Fue en este momento cuando nos percatamos de que, aunque nos mostraba todos aquellos atributos que necesitabamos, el algoritmo empleado no finalizaba cuando llegaba a un estado de parada o estado

final. Para poder conocer la función a modificar analizamos las clases y funciones de la librería que posee la siguiente estructura:

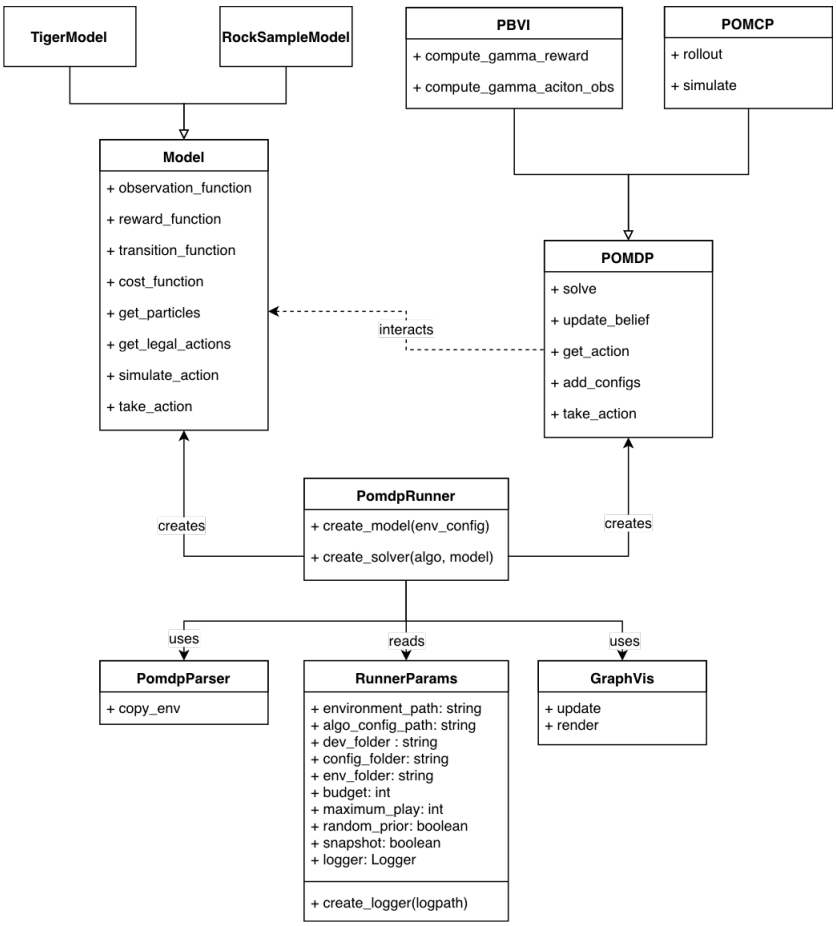

Figure 2: Estructura de la librería

Una vez analizada, realizamos la implementación y en este documento mostraremos la metodología empleada para detener el algoritmo cuando alcance uno de los estados finales. Además, también mostraremos cómo hemos implementado la simulación interactiva y silenciosa.

3.1 Problema del tigre

Para detener el problema del tigre simplemente era necesario indicar a la librería cual era la acción final. En este caso, cuando el sujeto abre la puerta, independientemente del lugar en el que se encuentre el tigre el problema finaliza ya que o bien el sujeto escapa o bien el tigre se come al sujeto. Para ello, identificamos en la librería la clase "pomdp_runner.py", que iteraba sobre los pasos y en concreto la función "run" que ejecutaba cada uno de ellos. Una vez aquí, introducimos el siguiente fragmento de

código:

```
if problema tigre then
    if accion abrir then
        muestra acción tomada final;
        muestra observación final;
        muestra recompensa final;
        para simulación;
    end
    muestra acción tomada;
    muestra nueva observación;
    muestra nueva recompensa;
    muestra nueva creencia;
end
```

Algorithm 1: Parada problema del tigre

Bastaba con comprobar si el problema elegido (valor del atributo "environment") era el problema del tigre y en caso de que la accion fuese abrir ("open") se mostrarían los parámetros: (1) Acción final, (2) Observación y (3) Recompensa. En otro caso, el algoritmo mostraría también los parámetros anteriores además de (4) Nueva creencia o *belief*.

3.2 Problema del tag

En este caso, para detener el problema del tag simplemente era necesario indicar cual era el estado final. Cuando el estado indica que el robot T ha sido atrapado el problema finaliza ya que significaría que el robot S ha atrapado al robot T. Para ello, en la clase "pomdp_runner.py" y en concreto en la función "run" comentada anteriormente introducimos el siguiente fragmento de código:

```
if problema tag then
    if estado actual tagged then
        muestra acción tomada final;
        muestra observación final;
        muestra recompensa final;
        para simulación;
    end
    muestra acción tomada;
    muestra nueva observación;
    muestra nueva recompensa;
    muestra nuevo estado;
end
```

Algorithm 2: Parada problema del tag

Bastaba con comprobar si el problema elegido (valor del atributo "environment") era el problema del tag y en caso de que el estado fuese atrapado ("tagged") se mostrarían los parámetros: (1) Acción final, (2) Observación y (3) Recompensa. En otro caso, el algoritmo mostraría también los parámetros

anteriores además de (4) Nuevo estado. El parámetro (5) Nueva creencia o *belief* decidimos omitirlo debido a que aparecían todos los valores de cada uno de los posibles estados en la consola y los resultados eran ilegibles prácticamente.

Sin embargo, leyendo la traza de la solución y observando cada uno de los pasos nos dimos cuenta de que, aunque el robot se encontrase en alguna de las casillas que formaban el límite del tablero, podía tomar como acciones aquellas que le permitirían salirse del mismo, es decir, acciones inválidas para ese estado como el ejemplo que aparece a continuación:

```
# Step = 648
*** 35 random particles are added ***
Taking action: East
Observation: Orv0rh7
Reward: -1.0
New state: Srv0rh7tv4th5
==================
# Step = 649
*** 35 random particles are added ***
Taking action: North
Observation: Orv0rh7
Reward: -1.0
New state: Srv0rh7tv4th5
```

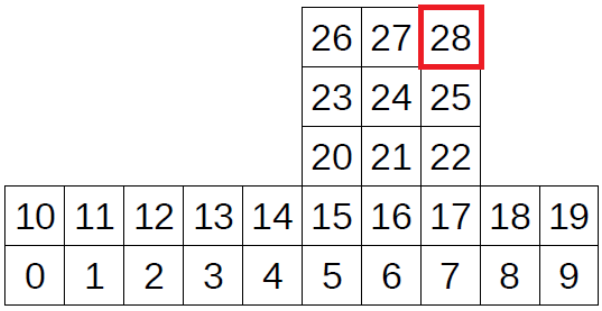

Figure 3: Acción inválida

En el ejemplo podemos apreciar que, aunque el robot S se encuentre en la casilla 28 (estado "Srv0rh7tv4th5") puede tomar la acción norte ("North") la cual es inválida ya que se saldría de los límites del tablero. Por ello, decidimos implementar la comprobación adjuntada en el Anexo A que para cada casilla límite del tablero restringiese las acciones que podría tomar el algoritmo. Esta comprobación se implementó en la función "get_legal_actions" de la clase "model.py" pero no se obtuvo éxito tras intentarlo en muchas ocasiones ya que el algoritmo llamaba un número muy alto de veces a esta función y no lo interpretaba de forma correcta.

3.3 Problema de los anuncios web

Para detener el problema de los anuncios web descrito en el Anexo B simplemente era necesario indicar cual era la acción final. Cuando la acción realizada es anunciar independientemente de qué se anuncie el problema finalizaría. Para ello, en la clase "pomdp_runner.py" y en concreto en la función "run" comentada anteriormente introducimos el siguiente fragmento de código:

```
if problema web then
    if accion anunciar then
        muestra acción tomada final;
        muestra observación final;
        muestra recompensa final;
        para simulación;
    end
    muestra acción tomada;
    muestra nueva observación;
    muestra nueva recompensa;
    muestra nueva creencia;
end
```

Algorithm 3: Parada problema de los anuncios web

Bastaba con comprobar si el problema elegido (valor del atributo "environment") era el problema de los anuncios web ("Web.POMDP") y en caso de que la acción fuese anunciar ("adv") se mostrarían los parámetros: (1) Acción final, (2) Observación y (3) Recompensa. En otro caso, el algoritmo mostraría también los parámetros anteriores además de (4) Nueva creencia o *belief*.

3.4 Problema de las islas

Por último, para detener el problema de las islas descrito en el Anexo C simplemente era necesario indicar cual era la acción final. Cuando la acción realizada es atracar el barco en alguna de las islas el problema finalizaría. Para ello, en la clase "pomdp_runner.py" y en concreto en la función "run" comentada anteriormente introducimos el siguiente fragmento de código:

```
if problema islas then
    if accion atracarr then
        muestra acción tomada final;
        muestra observación final;
        muestra recompensa final;
        para simulación;
    end
    muestra acción tomada;
    muestra nueva observación;
    muestra nueva recompensa;
    muestra nueva creencia;
end
```

Algorithm 4: Parada problema de las islas

Bastaba con comprobar si el problema elegido (valor del atributo "environment") era el problema de las islas ("Islands.POMDP") y en caso de que la acción fuese atracar ("arrive") se mostrarían los parámetros: (1) Acción final, (2) Observación y (3) Recompensa. En otro caso, el algoritmo mostraría también los parámetros anteriores además de (4) Nueva creencia o *belief*.

3.5 Simulación interactiva, silenciosa o benchmark

Tras detener el algoritmo en cada caso implementamos la posibilidad de elegir entre simulación interactiva o silenciosa. La librería usada ya mostraba por defecto en la salida el estado paso a paso por lo que únicamente tuvimos que implementar la simulación silenciosa.

Ante esta situación, decidimos que, si se utilizaba la instrucción original por defecto se ejecutaría la simulación interactiva y si ejecutábamos la instrucción original y además añadíamos un nuevo parámetro llamado "benchmark" seguido del número de simulaciones deseadas "X", se ejecutaría la simulación silenciosa un número X de veces como aparece a continuación:

>python main.py pomcp –env Tiger-2D.POMDP –**benchmark X**

Para ello, introducimos este nuevo parámetro en la clase "main.py" y en la función "__init__" de la clase "RunnerParams.py". Era necesario crear una nueva función llamada "runBench" en la clase "pomdp_runner.py" que ejecutase los pasos necesarios para resolver cada situación (de la misma forma que lo hacía la funcion "run" de la misma clase) pero esta vez sin mostrar los atributos de cada paso. Además, esta función almacenaría, cuando llegase a un estado final, en dos arrays llamados "step_list" y "fReward_list" el numero de pasos y la recompensa acumulada de cada una de las simulaciones para obtener las respectivas desviaciones típicas utilizando la función "statistics.stdev" de Python. De la misma forma, también almacenaría en los atributos "steps" y "fReward" el número total de pasos y la recompensa acumulada de todas las simulaciones con el objetivo de calcular posteriormente los valores medios.

Para mostrar finalmente los resultados de la simulación silenciosa introducimos en la clase "main.py" el siguiente fragmento de código:

if *interactiva* **then**
| ejecuta una simulación interactiva;
else
| **for** *X simulaciones* **do**
| | ejecuta X simulaciones;
| **end**
| muestra nº total de pasos;
| muestra recompensa total acumulada;
| muestra media de pasos;
| muestra media de recompensa;
| muestra desviación típica de los pasos;
| muestra desviación típica de la recompensa;
end

Algorithm 5: Muestra de los resultados

4 Resultados

A continuación, se muestran los resultados obtenidos tras ejecutar los siguientes casos para cada problema. La columna "Silenciosa X" representa el número (X) de simulaciones silenciosas que se han realizado. En esta columna el valor que aparece para la fila "Pasos" y "Recompensa" representa el valor total de pasos realizados y la recompensa acumulada total en dicha batería.

4.1 Problema del tigre

	POMCP		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	4	2	113
Recompensa	7.0	-101.0	107.0
Valor medio de pasos			3.76
Valor medio de recompensa			3.56
Desviación típica de pasos			2.17
Desviación típica de recompensa			19.67

	PBVI		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	5	3	112
Recompensa	6.0	8.0	64.0
Valor medio de pasos			3.73
Valor medio de recompensa			2.13
Desviación típica de pasos			1.87
Desviación típica de recompensa			21.38

4.2 Problema del tag

	POMCP		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 15
Pasos	5	4	1383
Recompensa	6.0	-4.0	-1218.0
Valor medio de pasos			92.2
Valor medio de recompensa			-81.2
Desviación típica de pasos			82.79
Desviación típica de recompensa			82.79

Para este problema hemos decidido mostrar tan solo 15 simulaciones silenciosas debido al numeroso tiempo que necesita para alcanzar el estado final por el problema de las acciones citado en el apartado 3.2. Además, también nos ha resultado imposible ejecutar el algoritmo PBVI sobre este problema ya que no toma ninguna acción ni muestra ningún error en la consola, simplemente no hace nada. Esta cuestión fue al comentada al profesor en tutoría y nos aclaró que es debido a que la implementación de

PBVI de la que disponemos es incapaz de resolverlo. En base a nuestra investigación [4], el algoritmo PBVI debería ser capaz de tratar con problemas de gran dimensión y resolverlos de forma más eficiente que la mayoría de algoritmos resolutivos.

4.3 Problema de los anuncios web

	POMCP		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	2	2	87
Recompensa	1.9	-4.7	21.5
Valor medio de pasos			2.9
Valor medio de recompensa			0.71
Desviación típica de pasos			1.49
Desviación típica de recompensa			1.91

	PBVI		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	5	3	76
Recompensa	6.0	8.0	12.4
Valor medio de pasos			2.53
Valor medio de recompensa			0.41
Desviación típica de pasos			0.82
Desviación típica de recompensa			2.37

4.4 Problema de las islas

	POMCP		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	5	4	177
Recompensa	96.0	-4.0	2754.0
Valor medio de pasos			5.9
Valor medio de recompensa			91.8
Desviación típica de pasos			2.14
Desviación típica de recompensa			18.22

	PBVI		
	Interactiva	Interactiva con 4 jugadas	Silenciosa 30
Pasos	6	4	73
Recompensa	95.0	-4.0	39.0
Valor medio de pasos			2.43
Valor medio de recompensa			1.3
Desviación típica de pasos			4.68
Desviación típica de recompensa			316.6

5 Conclusiones

Tras la realización del trabajo completo y a la vista de lo resultado obtenidos, hemos observado que todos los problemas que se ejecutan con el algoritmo PBVI se resuelven más rápido que con el algoritmo POMCP. Sin embargo, podemos apreciar en los resultados que el algoritmo POMCP obtiene una mayor recompensa si ejecutamos un amplio número de simulaciones.

A la hora de hacer pruebas con el problema tag y la creación del cuarto problema nos dimos cuenta de que las acciones las toma en función de la recompensa o *reward*, cosa que parece lógica, pero nosotros pensábamos que las acciones las tomaba en base a probabilidad partiendo del estado inicial y posteriormente tomaba la *reward* asociada a dicho movimiento. Podríamos penalizar enormemente los movimientos "ilegales" del problema del tag de esta forma, pero supone un gasto de tiempo del que no disponemos ya que hay que contemplar para cada estado "inicial" los posibles movimientos ilegales, todos los estados finales a los que se podría llegar para penalizar enormemente dichos movimientos ilegales. Lo cual implicaría desglosar todas y cada una de esas posibilidades. Teniendo en cuenta la cantidad de estados que tiene el problema del tag, es algo inviable para nosotros actualmente. No obstante, podemos perfeccionar el problema de las islas (diseñado por nosotros) debido a esto, ya que, inicialmente siempre lo resolvía con uno o dos movimientos pero tras darnos cuenta de lo mencionado anteriormente, ha aumentado el numero de pasos y por tanto el "realismo" del problema.

Como conclusión final, podríamos decir que ya que todo este entorno era desconocido para nosotros: tanto el lenguaje de programación, como el uso de librerías de Python y el funcionamiento de estos algoritmos; sentimos que el esfuerzo dedicado a este proyecto nos ha ayudado a ser más creativos en la resolución problemas que nunca antes habíamos afrontado.

6 Referencias

[1] PyPOMDP. Di, Lu. Data science software engineer at the University of Sydney.
`https://github.com/namoshizun/PyPOMDP`

[2] Pycharm, Jet Brains.
`https://www.jetbrains.com/pycharm`

[3] PyPOMDP. Luis Candelario Luna.
`https://github.com/LuisCande/PyPOMDP`

[4] Point-based value iteration: An anytime algorithm for POMDPs. Joelle Pineau, Geoff Gordon and Sebastian Thrun. Carnegie Mellon University
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.8961&rep=rep1&type=pdf`

7 Anexos

7.1 Anexo A: Comprobación de acciones válidas

```
if 'Srv' in state:
    # Getting the column and row
    vertical = state[3:4]
    horiz = state[6:7]

    # FIRST ROW
    # If the robot A is in the first row of the map, he
    # shouldn't be able to move to the North
    f_actions = self.actions
       if vertical == 0:
            f_actions.remove('North')
            # Cell 26, cant go north nor west
            if horiz == 5:
                f_actions.remove('West')
            # Cell 28, cant go north nor east
            if horiz == 7:
                f_actions.remove('East')

    # SECOND ROW
    # If the robot A is in the second row may not be
    # able to go west or east, depending on the cell
        if vertical == 1:
            # Cell 23, cant go west
            if horiz == 5:
                f_actions.remove('West')
            # Cell 25, cant go east
            if horiz == 7:
                f_actions.remove('East')

    # THIRD ROW
    # If the robot A is in the third row may not
    # be able to go west or east, depending on the cell
        if vertical == 2:
            # Cell 20, cant go west
            if horiz == 5:
                f_actions.remove('West')
            # Cell 22, cant go east
            if horiz == 7:
                f_actions.remove('East')

    # FOURTH ROW
    # If the robot A is in the fourth row may not be
```

```
# able to go north, west or east, depending on the
# cell
    if vertical == 3:
        # Cells 10, 11, 12, 13, 14, 18 and 19 can't
        # go north
        if horiz == 0 or horiz == 1 or horiz == 2
        or horiz == 3 or horiz == 4 or horiz == 8
        or horiz == 9:
            f_actions.remove('North')
            # Cell 10, can't go west either
            if horiz == 0:
                f_actions.remove('West')
            # Cell 19, can't go east either
            if horiz == 9:
                f_actions.remove('East')

# FIFTH ROW / LAST ROW
# If the robot A is in the last row of the map,
# he shouldn't be able to move to the South
    if vertical == 4:
        f_actions.remove('South')
        # Cell 0, can't go west either
        if horiz == 0:
            f_actions.remove('West')

        # Cell 10, can't go east either
        if horiz == 9:
            f_actions.remove('East')

    return f_actions
```

7.2 Anexo B: Problema de los anuncios web

Disponemos de una web que ofrece anuncios de corbatas o skates y esta es visitada por jóvenes y ejecutivos. El 70% de los usuarios que visitan nuestra web son jóvenes y el 30% ejecutivos. Para poder seguir manteniendo nuestra web necesitamos anunciar el producto adecuado en función del usuario que visite nuestra web. Asumimos que si anunciamos una corbata a un ejecutivo directamente realizará la compra al igual que sucedería con los skates y los jóvenes y en caso de que fallemos, no comprarán nuestro producto. Sin embargo, no conocemos realmente qué perfil está visitando nuestra web y tendremos que observar las consultas que realizan, que pueden ser búsquedas de páginas sobre golf o sobre videojuegos respectivamente para acertar con nuestro anuncio.

7.3 Anexo C: Problema de las islas

Imagina que eres el capitán de un barco pirata. En la taberna del último pueblo que visitásteis, te hicieron llegar el mapa del tesoro más grande jamás visto. La localización del mapa se corresponde con un archipiélago de tres islas. Cada una de las islas tiene sus peculiaridades: una de ellas esconde el tan afamado tesoro, otra de ellas un pueblo donde descansar para la siguiente aventura y la última está maldita, en ella habitan seres del inframundo tan hostiles como se pueda imaginar. Como capitán del barco puedes ordenar atracar en cualquiera de las tres islas y exponerte a lo que en ellas te aguarde o bien puedes ordenar a un miembro de la tripulación usar el catalejo para obtener información de las islas y así tomar una decisión más segura a la hora de atracar.