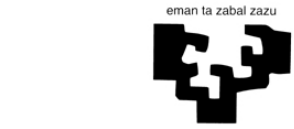



ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

# Villapresenteko labirintoa: errefortzuzko ikasketa bidezko ebazpena

Gradu Amaierako Lana
Matematikako Gradua

Ane Matabuena Zugasti

Inmaculada Arostegi Madariaga
Urtzi Ayesta Morate
Irakasleek zuzendutako lana

Leioa, 2021ko ekainaren 18a

# Aurkibidea

# Sarrera

Adimen artifizialaren munduarekiko jakin-mina gradu amaierako tutoreak lana errefortzuzko ikaskuntza bitartez lantzea proposatu zidanean hasi zen. Ordura arte aipatutako gaia ez nuen ezagutzen, ondorioz, errefortzuzko ikaskuntzari buruzko hainbat liburu eta artikulu irakurri behar izan ditut, baita horri buruzko bideo ugari ikusi ere.

Gaia landuta eta errefortzuzko ikaskuntzaren bitartez ebatz daitezkeen hainbat problema aztertuta, lana labirintoen mundura bideratzea erabaki nuen.

Gainera, unibertsitateko prestakuntza-urteetan bereganatutako ezagutza adimen artifiziala bezalako gai batean islatzea interesgarria iruditzen zitzaidan, orain arte niretzat gai horiek zientzia-fikzioa baino ez ziren eta.

Hasiera batean, lanaren oinarrizko planteamendua fikziozko labirinto bat diseinatzea zen bertatik irteteko bide optimoa aurkitzearren. Hala ere, azkenean, bilaketa lan neketsu baten ostean, benetako labirinto aproposa aurkitu nuen Kantabriako Villapresente udalerrian (43 ° 21 '55.0 "n 4 ° 07 '03.0" W). Ondorioz, aipatutako labirintoa lanaren atal praktikoaren oinarria bilakatzea erabaki nuen.

Villapresenteko labirintoa 2017an sortu zuen Emilio Pérez-ek eta haren eraikuntzan bi metro eta erdiko 40.000 pinu erabili zituen. Labirintoak 5.623 metro karratu ditu, ondorioz, Espainiako labirinto handiena da. Bitxikeria gisa aipa daiteke, haren irudia ia karratu bat dela, eta sarrera eta irteera alde berean daudela landare-hormen bitartez bereizita.

Labirinto horretatik irteteko ibilbide optimoa bilatzea sari sakabanatuen problema gisa planteatu da, amaierako laukia saria eskaintzen duen bakarra da eta. Planteamendu horrek zailtzen du labirintotik irteteko ibilbide optimoa errefortzuzko ikaskuntzaren bidez lortzea, amaierako laukira iritsi arte ez baita inolako saririk lortzen.

Errefortzuzko ikaskuntzaren ezaugarriak eta labirintoaren dimentsioa kon-

tuan hartuta, hasiera batean labirintotik irteteko ibilbide optimoaren bilaketaren planteamendua labirintoari buruzko informazio guztia genuela suposatuz landu zen.

Lana aurrera egin ahala, beste planteamendu bat landu zen. Kasu horretan, eragilea eta ingurunea sartzen dira jokoan. Eragileak ez du ingurunea ezagutzen, eta labirintoaren bide egokiena aurkitzeko, iraganean ikasitakoa esplotatu behar du, baita ekintza berriak esploratu ere. Esplotazioa eta esplorazioa errefortzuzko ikaskuntzaren oinarriak dira.

Deskribatutako planteamendua kontuan hartuta, lana jarraian azaltzen diren zatitan eta kapituluetan egituratu da.

Lanaren lehenengo zatian labirintoaren bide optimoa bilatzeko aplikatutako metodoak ulertzera eramango gaituen marko teorikoa azaltzen da. Azkenengo zatian, ordea, metodo horien aplikazioaren emaitzak ez ezik, lortutako ondorioak ere islatzen dira.

Hain zuzen ere, lanaren lehenengo kapituluan Markov-en erabaki-prozesuak (MDP) zer diren azaltzen da. Prozesu horiei esker, problema modeliza daiteke, gero harekin lan egin ahal izateko. Metodo hori oinarri moduan hartu da problemaren informazio guztia ezagutzen dela suposatzen den egoeran ebazteko. Gainera, kapitulu horretan bertan, prozesu horiekin lotutako beste kontzeptu batzuk ere landu dira.

Bigarren kapituluan labirintotik irteteko bide optimoa lortzeko erabilitako metodoa (balio-iterazioa) azaltzen da, problemaren informazio guztia erabat ezagutzen dela kontuan hartuta.

Hirugarren kapituluan jada errefortzuzko ikaskuntzaren munduan murgilduko gara. Kasu horretan, ingurune ezezaguna duen MDP bat lantzen da. Alegia, ez da problemaren informazio guztia ezagutzen. Baldintza horietan esplotazio eta esplorazioaren bidez problema ebazteko erabilitako metodoa azaltzen da (Q-learning).

Azkenik, laugarren kapituluan, aurreko kapituluetan azaldutako marko teoriko osoa praktikan jartzen da. Horrela, labirintotik irteteko biderik optimoa lortu nahi izan da bi egoera desberdinen aurrean: problemaren informazio osoa dugunean, eta, ordea, eragileak problemaren ingurunea ezagutzen ez duenean. Atal praktiko horretan lortutako emaitzek bi metodo hauen konbergentzia sari sakabanatuetako dimentsio handiko problemetetan aztertzea ahalbideratu digute.

# 1. kapitulua

# MDP: Markov-en erabaki-prozesua

Sarreran esan den bezala, kapitulu honetan Markov-en erabaki-prozesuak azalduko ditugu. Labirintoaren ibilbide laburrena aurkitzeko, erabakiak hartu behar dira bidean zehar. Hau da, kontrol optimoa ebatzi behar da. Hortaz, prozesu horien bitartez problemak modelizatu ahal dira, ondoren, problemei soluzioa emateko. Azalduko den hurrengo atala ulertzea garrantzitsua da, errefortzuzko ikaskuntza ulertzera eramango gaituen oinarriak biltzen dituelako.

Hurrengo hiru kapituluak garatzeko David Silver-ek egindako kurtso onlineko materiala [2], baita Richard S. Sutton-ek eta Andrew G. Barto-k idatzitako *Reinforcement Learning: An Introduction* [8] liburuko kontzeptuak ere erabili dira gehienbat.

## 1.1 Markov-en erabaki-prozesuaren definizioa

Markov-en erabaki-prozesuak denbora diskretuan lan egiten duten kontrol estokastikoko prozesuak dira. Problema baten egoera guztiek Markov-en propietatea betetzen badute, problema prozesu horren bitartez modeliza daiteke.

**Markov-en propietatea.** Markov-en propietateak adierazten du etorkizuna independientea dela orainaldian emandako iraganarekin. Hau da:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{1.1}$$

Hortaz, $S_t$ egoerak aurreko egoera guztien informazioa biltzen du, eta beraz, aurreko egoera bakoitzaren informazioa ez da baliogarria [7].

**Definizioa 1.1.1. Markov-en erabaki-prozesua(MDP).** Markov-en erabaki-prozesua $\omega =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ tuplaren bidez definitzen da. Tupla osatzen duten elementuak hurrengoak dira:

- $\mathcal{S}$: Egoeren multzo finitua da.
- $\mathcal{A}$: Akzioen multzo finitua da.
- $\mathcal{P}$: $\mathcal{S}$ x $\mathcal{A} \to \mathbb{P}(\mathcal{S})$ probabilitate-banaketa baten moduan adierazitako egoeren arteko trantsizio-probabilitate funtzioa da.
  $\mathcal{P}^a_{ss'}$-k, $t$ denbora-urratsean $s \in \mathcal{S}$ egoeratik $a \in \mathcal{A}$ akzioa eginez, $t+1$ denbora-urratsean $s' \in \mathcal{S}$ egoerara ailegatzeko probabilitatea adierazten du. Hau da:

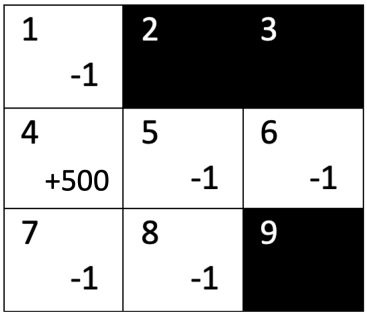
$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a].$$

- $\mathcal{R}$: $\mathcal{S}$ x $\mathcal{A} \to \mathbb{R}$ sari-funtzioa da.
  $\mathcal{R}^a_s$-k, $t$ denbora-urratsean $s \in \mathcal{S}$ egoeratik $a \in \mathcal{A}$ akzioa eginez, $t+1$ denbora-urratsean esperotako saria adierazten du. Hau da:

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a].$$

- $\gamma$: Deskontu-faktorea da eta 0 eta 1 artean definituta dago. Hurrengo atalean kontzeptu horretan sakonduko dugu.

Behin bost elementu horiek azalduta, elementu bakoitza hobetu ulertzeko, labirintoaren adibidearekin lan egingo dugu. $\gamma$ parametroa aurrerago aztertuko dugu. Demagun hurrengo labirintoaren eremua daukagula*:




**1.1. irudia**: Labirintoaren eremu bat (asmatuta).

Labirintoan egoeren multzo finitua posizio libre guztiak izango dira; hots, 1,4,5,6,7 eta 8 laukiak. Gainera, lau akzio posible finkatuko ditugu: 'gora

*Lauki bakoitzaren goiko zenbakiak egoera etiketatzen du eta beheko zenbakiak egoera bakoitzaren saria adierazten du.

joan, 'behera joan', 'eskumara joan' eta 'ezkerrera joan'. Hortaz egoera batetik bestera mugitzeko, nahitaez aurreko akzioren bat erabili behar da.

Bestalde, trantsizio-probabilitate funtzioak adieraziko du $s$ egoeratik, akzio konkretu bat hartuz, beste egoerara joateko probabilitatea. Adibidez, 5. egoeratik, ezkerrera eginez, 4. egoerara joateko probabilitatea 1 izango da, baina 5. egoeratik, behera eginez, 4. egoerara joateko probabilitatea 0 izango da.

Azkenik, sari-funtzioak egoera batean akzio bat hartuta esperotako saria adierazten du. Adibidez, 5. egoeratik, ezkerrera eginez, +500 saria espero da. Alegia, 4. egoerara egin da trantsizioa eta bertan +500 saria dago finkatuta. 5. egoeratik behera eginez, aldiz, -1 saria espero da 8. egoerara egin delako trantsizioa.

Adibidea alde batera utzita, Markov-en erabaki-prozesuetan parte hartzen duen beste elementu garrantzitsu bat azalduko dugu: *politika*.

**Definizioa 1.1.2. Politika.** $s \in \mathcal{S}$ egoera emanda, akzioen probabilitate-banaketa itzultzen duen funtzioa da. Hots, $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$.

Politikek mapa itxurako egitura jarraitzen dute. Hain zuzen ere, bi politika mota bereiz daitezke. Izan bitez $s \in \mathcal{S}$ eta $a \in \mathcal{A}$, orduan:

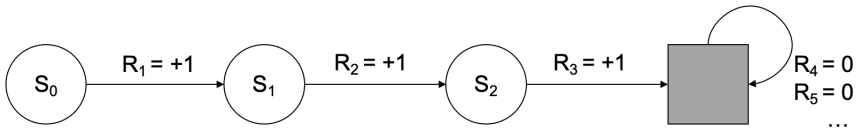
- **Politika determinista:** $a = \pi(s)$.
- **Politika estokastikoa:** $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$.

Sari totala adierazteko bi era ezberdin daude. Hala ere, zenbait baldintzak hartu behar dira kontuan adierazpen bat edo bestea erabiltzeko.

## 1.2 Sari totala

Sari totala definitu baino lehen, bi kontzeptu definitu behar ditugu: amaierako egoera eta gertaerak.

**Definizioa 1.2.1. Amaierako egoera.** Amaierako egoera (edo egoera xurgatzailea) egoera bat da non, 0 balioko sariarekin, akzio guztiak 1 probabilitatearekin egoera bera horretara igarotzen duten [3].




**1.2. irudia**: Amaierako egoera duen pauso-sekuentzia.

**Definizioa 1.2.2. Gertaera.** Hasierako egoera batean hasi eta amaierako egoera batean bukatzen den pauso-sekuentzia bat da [3].

Gerta daiteke hasierako egoera batetik hasita, pauso-sekuentzia amaierako egoera batean ez bukatzea. Horrelakoetan, pauso-sekuntzia horri **etengabeko lana** deitzen zaio [5].

Sari totaleko kontzeptura bueltatuz, esan bezala, bi adierazpen mota daude.

Lehenengo adierazpenari dagokionez, $t$ denbora-urrats ostean jasotako sariak $R_{t+1}$, $R_{t+2}$,... izanik, sari totalaren adierazpen sinpleena hurrengoa da:

$$G_t^* = R_{t+1} + R_{t+2} + ... + R_T, \tag{1.2}$$

non $T$ amaierako egoera den.

Hala ere, aurreko formulak arazoak sortzen ditu, ezin baitira kalkuluak egin $T$-k ez duenean limitirik. Ondorioz, sari totala bigarren adierazpen batekin defini daiteke: itzulera [5].

**Definizioa 1.2.3. Itzulera.** Itzulera $t$ denbora-urrats bakoitzean deskontutako sari totala da:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1.3}$$

non $\gamma$ deskontu-faktorea den.

**Definizioa 1.2.4. Deskontu-faktorea.** Deskontu-faktorea etorkizuneko sariaren oraingo balioa da. $\gamma$ sinboloaren bidez adierazten da eta $\gamma \in [0,1]$.

$\gamma$ parametroaren balioa 0 bada, garrantzia ematen zaio berehalako sariak maximizatzeari. $\gamma$-ren balioa 1 bada, aldiz, etorkizuneko sariei garrantzia handiagoa ematen zaie.

Markov-en erabaki-prozesuaren helburua itzulerak maximizatuko duen

$\pi$ politika aurkitzea da. Alegia, egoera batetik hasita, etorkizunean espero den sari totala maximizatzen duen politika topatzea. Politika $\pi$ hori aurkitzen badu, problema ebatzi duela esan nahi du.

Markov-en erabaki-prozesuetan eta sarien prozesu gehienetan itzulera erabiltzen da sari totala adierazteko. Adierazpen hori erabiltzea sustatzen duten arrazoien artean hurrengoak azpimarra daitezke, besteak beste:

- Eredu perfektua existitzen ez dela adierazten duelako.
- Matematikoki komenigarria delako.
- Markov-en prozesu ziklikoetan itzulera infinituak ekiditen dituelako.

Atal honekin bukatzeko, azpimarratzeko da $\gamma$=1 bakarrik erabil daitekeela gertaera guztiak amaierako egoeran bukatzen direnean.

## 1.3 Balio-funtzioa

Politika jakin baterako egoera batean egotea ona edo txarra den zehazteko, edo egoera batean akzio bat hartzea ona edo txarra den ebaluatzeko, balio-futzioak baliogarriak dira. Horiek hasierako egoeratik baldintzatuta, edo hasierako egoeratik eta hartutako akzioatik baldintzatuta, aurrean aipatutako itzulera erabiltzen dute.

Bi balio-funtzio ezberdin sailka daitezke: egoera-balio funtzioa eta akzio-balio funtzioa.

- **Egoera-balio funtzioa**: Egoera-balio funtzioak egoera baterako eta $\pi$ politika konkretu baterako, $s$ egoeratik hasita eta politika $\pi$ jarraituz espero den itzulera adierazten du. $V_\pi(s)$ moduan adierazten da eta hurrengo eran definitzen da:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \\ &= \mathbb{E}_\pi[G_t|S_t = s]. \end{aligned} \tag{1.4}$$

- **Akzio-balio funtzioa:** Akzio-balio funtzioak $\pi$ politika konkretu bat jarraituz, $s$ egoera batetik hasita $a$ akzioa egitean lortzen den esperotako itzulera adierazten du. $Q_\pi(s, a)$ moduan adierazten da eta

hurrengo eran definitzen da:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \end{aligned} \tag{1.5}$$

Egoera-balio funtzioa eta akzio-balio funtzioa baliokideak izan arren, aipatu behar da ikasketa algoritmoetan akzio-balio funtzioa erabili ohi dela.

## 1.4 Bellman-en itxaropen-ekuazioak

Balio-funtzioek erlazio errekurtsibo bat betetzen dute. Ezaugarri hori asko erabiltzen da hurrengo kapituluetan landuko ditugun problemen ebazpen metodoetan. Edozein $\pi$ politikarako eta edozein $s$ egoerarako, $s$ egoeraren eta egoera horren ondorengo egoeraren, $s'$-ren, arteko harremana defini daiteke.

Horretarako, lehenik eta behin, $t$ denbora-urratserako itzuleraren formula garatuko dugu:

$$\begin{aligned} G_t &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) = R_{t+1} + \sum_{k=0}^{\infty} \gamma^k R_{t+k} \\ &= R_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{1.6}$$

Beraz, aurreko adierazpena kontuan hartuz, $V_\pi$**-rako Bellman ekuazioa** hurrengoa da:

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]. \tag{1.7}$$

Eta $Q_\pi$**-rako Bellman ekuazioa** ondorengoa da:

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]. \tag{1.8}$$

Hala ere, $V_\pi$-rako Bellman ekuazioa, baita $Q_\pi$-rako Bellman ekuazioa ere, beste era batzuetan adieraz daitezke. Izan ere, 1.7 formulatik abiatuz eta *baldintzazko itxaropenaren teorema* erabiliz hartzen den akzioan baldintzatuz, hurrengo adierazpen hau lortzen da:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a). \tag{1.9}$$

Era berean, 1.8 formulatik abiatuz eta baldintzazko itxaropenaren teorema erabiliz hurrengo $s'$ egoeran baldintzatuz, hurrengo formula hau lortzen da:

$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s'). \tag{1.10}$$

Edo, aurreko bi formulak, 1.9 eta 1.10, kontuan hartuz:

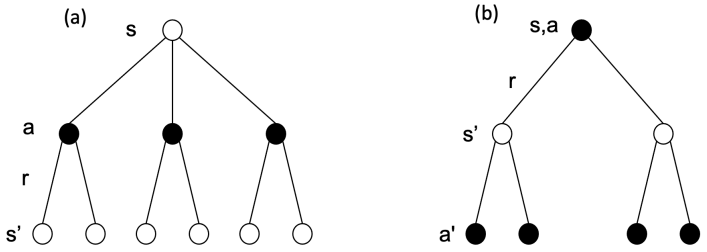
$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s')] \tag{1.11}$$

eta,

$$Q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a') \tag{1.12}$$

formulak lortu ahal ditugu.

Azkenengo bi formula horiek *backup*-diagramen [†] bitartez irudikatu ahal dira.




**1.3. irudia**: $V_\pi$-ren (a) eta $Q_\pi$-ren (b) *backup*-diagramak

Beraz, adierazitako azkenengo Bellman ekuazio horiek egoera-balio funtzioaren eta akzio-balio funtzioaren kalkulua sinplifikatzen dute.

## 1.5 Balio-funtzio optimoa

Politika deskribatzeko eta ebaluatzeko, egoera-balio funtzioa eta akzio-balio funtzioa erabiltzen dira. Elementu horiek hobetu ezin direnean, **optimoak** direla esango da.

[†] **Backup-diagramak** eguneratzeen oinarria osatzen duten erlazioak irudikatzen dituzten diagramei deritze.

- **Egoera-balio funtzio optimoa**: Politika guztietatik egoera-balio funtzio maximoa ematen duena izango da eta $V_*(s)$ moduan adieraziko da. Alegia,

$$V_*(s) = \max_\pi V_\pi(s). \tag{1.13}$$

  Hau da, egoera-balio funtzio optimoak $s$ egoeratik hasita lor daitekeen itzulera maximoa emango du.

- **Akzio-balio funtzio optimoa**: Politika guztietatik akzio-balio funtzio maximoa ematen duena izango da eta $Q_*(s,a)$ moduan adieraziko da:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a). \tag{1.14}$$

  Akzio-balio funtzio optimoak, aldiz, $s$ egoeratik hasita eta $a$ akzioa hartuz lor daitekeen itzulera maximoa emango du.

Beraz, oraingoan, egoera-balio funtzio optimoaren eta akzio-balio funtzio optimoaren adierazpen ezberdinak lortuko ditugu. Adierazpen horiek lortzeko, alde batetik, 1.9 eta 1.10 adierazpenak erabili ahal ditugu. Horrela,

$$V_*(s) = \max_{a\in\mathcal{A}} Q_*(s,a) \tag{1.15}$$

eta

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_*(s') \tag{1.16}$$

ekuazioak lortzen ditugu.

Beste aldetik, aurreko bi formulak erabiliz, beste adierazpenak lortu ahal ditugu:

$$V_*(s) = \max_{a\in\mathcal{A}} [\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_*(s')] \tag{1.17}$$

eta

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'\in\mathcal{A}} Q_*(s',a'). \tag{1.18}$$

Aurreko ekuazioei **Bellman-en itxaropen ekuazio optimoak** deritze.

## 1.6 Politika optimoa

Definizioz, politika bat bestea baino hobea izango da politika horren balio-funtzioa bigarren politikaren balio-funtzioa baino handiagoa bada egoera guztietarako. Alegia,

$$\pi \geq \pi' \text{ edozein } s \in \mathcal{S}\text{-rako } V_\pi(s) \geq V_{\pi'}(s) \text{ betetzen bada.}$$

Hurrengo teoremak adierazten digu nola lortu ahal dugun politika optimoa.

**Teorema 1.6.1.** *Edozein Markov-en erabaki-prozesurako:*

- *Existituko da beste edozein politiken baino edo bezain hobea den $\pi_*$ politika optimoa, $\pi_* \geq \pi \ \forall \pi$.*
- *Politika optimo guztiek egoera-balio funtzio optimoa lortzen dute: $V_{\pi_*}(s) = V_*(s) \forall s \in \mathcal{S}$.*
- *Politika optimo guztiek akzio-balio funtzio optimoa lortzen dute: $Q_{\pi_*}(s,a) = Q_*(s,a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.*

Teorema horren frogapena A eranskinean A.0.1 atalean frogatuta dago [6].

Beraz, bi aukera daude politika optimoa lortzeko. Alde batetik, politika hobetuz politika optimoa lor dezakegu. Beste aldetik, balio-funtzioak eguneratu ahal ditugu, balio optimoak eskuratu arte, eta ondoren politika optimoa lortu. Bi aukera hauek detaile gehiagorekin hurrengo atalean jorratuko ditugu.

# 2. kapitulua

# MDPk ebazteko metodoak

Aurreko kapituluan ikusi dugu nola modelizatu ahal dugun problema bat Markov-en erabaki-prozesua erabiliz. Problemaren informazio guztia ezagututa, problema MDP moduan adieraz daiteke eta prozesu horren elementu guztiak definitu ahal ditugu. Beraz, kapitulu honetan ikusiko dugu nola ebatzi ahal diren prozesu horren bitartez definitu ahal diren problemak programazio dinamikoa erabiliz. Garrantzitsua da argi izatea kapitulu honetan problemaren informazio osoaren ezagutzarekin lan egiten dugula.

## 2.1 Programazio dinamikoa

Balio-iterazio teknikan sakondu baino lehen **programazio dinamikoa** zer den azalduko dugu.

*Programazio dinamikoa* bi termino ezberdinetan bana daiteke: *programazioa* eta *dinamikoa*. Alde batetik, *dinamikoak* problemak osagai sekuentzialak edo tenporalak dituela adierazi nahi du. Beste aldetik, *programazioak* optimizazio politika bati egiten dio erreferentzia. Beraz, *programazio dinamikoa* problema dinamiko konplexuak ebazteko metodo bat da non problema azpiproblemetan banatzen den, eta azpiproblemak ebatziz eta konbinatuz problema osoa ebazten den [9].

Programazio dinamikoa erabili ahal izateko, kapituluaren sarreran esan dugun bezala, MDP osatzen duten elementu guztiak ezagutu behar dira. Izan ere, trantsizio-probabilitate funtzioa eta sari-funtzioa ezagutzea ezinbestekoa da. Beraz, baldintza hori betez gero, programazio dinamikoa aplika daiteke problema ebazteko.

Programazio dinamikoaren barnean hiru metodo sailkatu ahal ditugu: *politika-ebaluazio iteratiboa*, *politika-iterazioa* eta *balio-iterazioa*.

Alabaina, *politika-ebaluazio iteratiboa* eta *politika-iterazioa* errendimendua ebaluatzeko problemak ebazteko erabiltzen dira eta *balio-iterazioa*, aldiz, kontrolezko problemak ebazteko. Ikus dezagun zeintzuk diren bi problema horien arteko ezberdintasunak.

**Errendimendua ebaluatzeko problemetan**, Markov-en erabaki-prozesua ez ezik, politika $\pi$ ezagutzen dugu. Problema honen helburua politika ona edo txarra den ebaluatzea da. Horretarako, ezagutzen den $\pi$ politikarako balio-funtzioak estimatzen dira.

**Kontrolezko problemetan**, aldiz, Markov-en erabaki-prozesua ezagutzen dugu bakarrik. Hau da, ez digute $\pi$ politika bat ematen. Hortaz, kontrolezko problemetan, gure helburua politika optimoa lortzea da.

Gradu amaierako lan honen helburua Villapresenteko labirintutik irteteko modu eraginkorrena aurkitzea denez, kontrolezko problema bat dugu. Beraz, informazio guztia ezagutzen dugula, eta MDP definitzeko elementu guztiak aldez aurretik ondo zehaztu ahal ditugula suposatuz, programazio dinamikoren **balio-iterazioa** metodoa erabiliko dugu lanaren helburua lortzeko.

**Balio-iterazioa**

Balio-iterazio teknikaren bitartez era errekurtsiboan kalkula daitezke MDP baten egoeren balioak. Horretarako, esan den bezala, trantsizio-probabilitate funtzioa eta sari-funtzioa ezagutu behar dira nahitaez.

Balio-iterazio metodoak $V(s)$-ren balioespena iteratiboki hobetuz, $s$ egoeraren egoera-balio funtzio optimoa kalkulatzen du. Horretarako, edozein $k \in \mathbb{N}$-rako, Bellman-en itxaropen ekuazio optimoa erabiliz:

$$V_{k+1} \leftarrow \max_{a \in \mathcal{A}} (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s')) \tag{2.1}$$

adierazpena lortzen da.

Egoera guztien balio optimoa lortzeko, lehendabizi, metodo honek egoera guztien balioak zoriz hasieratzen ditu. Ondoren, aurreko formula erabiltzen du egoera bakoitzerako balio-funtzio berri bat lortzeko. Prozesua behin eta berriz errepikatuz, metodoak egoera bakoitzaren balio-funtzioa hobetuko du, optimora konbergitu arte. Era horretan, problemaren egoera guztien politika optimoa lortuko da.

**Teorema 2.1.1.** *Balio-iterazioak egoera-balio funtzio optimora konbergitzen du,* $V(s) \rightarrow V_*(s)\ \forall s \in \mathcal{S}$.

Teorema horren frogapena A eranskinean A.0.3 atalean dago [10].

Beraz, teoremaren arabera, balio-iterazioak, formalki, iterazio infinituak behar ditu balio optimora konbergitzeko. Hala ere, praktikoki, prozesua noiz bukatu behar den zehazteko metodo ezberdinak daude. Metodo horietako bat izan daiteke prozesua errepikatzea $n$ iterazioko eta $n+1$ iterazioko balio-funtzioen kenketaren maximoa $\theta$ baino txikiagoa izan arte (esate baterako $\theta = 0.001$).

Funtsean, esandakoa kontuan hartuz, balio-iterazioaren algoritmoa hurrengoa izango da:

**Balio-iterazioa algoritmoa**

Hasieratu $V(s) \forall s \in \mathcal{S}$ nahierara
**errepikatu iterazio bakoitzeko**
    $s \in \mathcal{S}$ **guztietarako egin**
        $a \in \mathcal{A}$ **guztietarako egin**
            $Q(s,a) \leftarrow \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s')$
        **bukatu**
        $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
    **bukatu**

# 3. kapitulua

# Errefortzuzko ikaskuntza

Kapitulu honetan eragilearen eta ingurunearen elementuekin lan egiten hasiko gara. Bi elementu berri horiek errefortzuzko ikaskuntzaren benetako funtsa dira. Aurreko bi kapituluetan, suposatu dugu problemaren informazio osoa ezagutzen genuela, hots, ikuspegi *globala* geneukala. Errefortzuzko ikaskuntzan, aldiz, eragileak ez du ingurune osoa ezagutuko. Ezberdintasun horrek guztiz ezaugarrituko du errefortzuzko ikaskuntza.

Oraingo baldintzetan, eragileak ez du labirintoaren informaziorik eta non dagoen baino ez daki. Hortaz, akzioak hartuz eta akzio horrek eragiten duen sariak jasoz, eragileak inguruneari buruz ikasi beharko du. Kapitulu honetan ikusiko dugu nola ebatzi ezaugarri horiek dituzten problemak errefortzuzko ikaskuntza erabiliz.

Villapresenteko labirintoaren ibilbide optimoa lortzeko, eragileak esplotazio eta esplorazio ekintzez baliatuko da. Alegia, ikasitakoa esplotatuko du eta akzio ezberdinak esploratuko ditu.

## 3.1 Errefortzuzko ikaskuntzaren osagaiak

Errefortzuzko ikaskuntza eragile baten eta ingurune baten arteko interakzioan oinarritzen da [1].

**Definizioa 3.1.1. Eragilea**. Errefortzuzko ikaskuntzaren ikastuna eta erabakiak hartzen dituen arduraduna da.

**Definizioa 3.1.2. Ingurunea**. Eragilea mugitzen den eta lan egiten duen lekua da. Une bakoitzean egon daitezkeen arauak eta mugak jasotzen ditu.

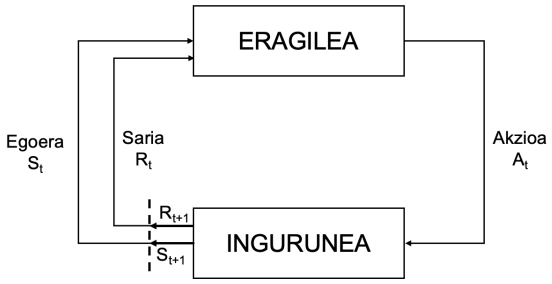
Momentu orotan bi osagai horien artean $feedback$-a dago. Berrelikadura horretan lan egiten duten elementuak hiru dira: egoera, akzioa eta saria.

**Definizioa 3.1.3. Egoera.** Denbora-urrats konkretu batean eragileak jasotzen duen ingurunearen irudikapena.

**Definizioa 3.1.4. Akzioa.** Momentu zehatz batean eragileak aukera dezakeen ekintza da.

**Definizioa 3.1.5. Saria.** Eragileak une zehatz batean egiten duen lanaren zuzentasun eta egokitasun maila islatzen duen eskalarra da *.

Beraz, eragileak eta inguruneak haien artean interakzionatzen dute denbora-urrats diskretuan ($t = 0, 1, 2, 3, ...$). $t$ urrats bakoitzeko, eragileak ingurunearen $S_t$ egoera jasotzen du. Ondoren, posiblea den akzio bat aukeratzen du $A_t \in \mathcal{A}$. Urrats bat geroago, eragileak $R_{t+1}$ saria jasoko du bere akzioaren ondorio gisa, baita hurrengo egoerara mugituko da ere ($S_{t+1}$). Eragilearen helburua izango da urrats bakoitzean akzio on bat aukeratzea etorkizuneko sari totala maximizatzeko. Ikusi 3.1. irudia.




**3.1. irudia**: Eragilea-ingurunea interakzioa errefortzuzko ikaskuntzan.

Labirintoaren kasuan, eragileak $s \in \mathcal{S}$ egoera batean egonda, $a \in \mathcal{A}$ akzio bat aukeratuko du. Hau da, labintoaren posizio (edo lauki) batean egonda, akzio posible guztietatik bat egitea erabakiko du. Beraz, eragilea $s' \in \mathcal{S}$ egoerara mugituko da, eta gainera, inguruneak sari bat emango dio $s \in \mathcal{S}$ egoeratik $a \in \mathcal{A}$ akzioa aukeratzeagarik. Sari horiek beti izango dira -1, eragilea amaierako egoerara ez ailegatu arte. Amaierako egoerara heltzean inguruneak +500 sariarekin sarituko du eragileak. Urratsez urrats, eragileak aurreko eskemaren prozesua errepikatuko du.

Errefortzuzko ikaskuntzaren osagaien atalarekin bukatzeko, azkenengo kontzeptu bat definituko ditugu: historia.

**Definizioa 3.1.6. Historia.** Egoeren, akzioen eta sarien seguida da.

Hots, $t$ urratsera arte gertatu diren elementu behagarri guztiak biltzen dituen seguida da.

*Saria ez da beti ona izan behar. Eragileak txarto joka dezake $t$ urrats batean eta jasotzen duen saria txarra izan daiteke. Horrelakoetan, **zigorra** moduan ezagutzen dugu saria

Beraz, ohartzen gara, aurreko kapituluetan azaldutako kontzeptu batzuk errepikatzen direla errefortzuzko ikaskuntzan. Alabaina, ikaskuntza honetan eragileak eta inguruneak parte hartzen dute, eta ingurunea guztiz ezezaguna da eragilearentzat. Ezaugarri horietako problema bat **Markov-en erabaki-prozesu ezezagun** baten bitartez defini daiteke, horietan trantsizio-funtzioa eta sari-funtzioa ezezagunak baitira.

## 3.2 Errefortzuzko ikaskuntzaren bidez problemak ebazteko metodoak

Eragilearen eta ingurunearen arteko interakzioetan oinarritzen diren problemak ebazteko bi era ezberdin daude. Esan bezala, eragileak ez du ingurunea ezagutzen, eta erabaki optimoa aurkitu nahi du. Horretarako ingurunearekin interakzionatu behar du. Hala ere, eragilearen baldintzen arabera, errendimendua ebaluatzeko problema eta kontrolezko problema baten aurrean egon gaitezke.

Alde batetik, demagun errendimendua ebaluatzeko problema bat daukagula. Hau da, demagun eragileak $\pi$ politika erabiltzen duela ingurunetik mugitzeko. $\pi$ politikaren balio-funtzioak estimatzeko, *Monte-Carlo* edo *TD-learning* teknikak erabil daitezke.

Beste aldetik, demagun kontrolezko problema bat daukagula. Hots, eragileak ez du $\pi$ politikarik ezagutzen. Beraz, gure helburua politika onena zein den aurkitzea da. Eragileak, ingurunea ezagutzen ez duenez, politika optimoa lortzeko *SARSA* edo *Q-learning* erabil ditzake.

Lehen esan dugun bezala, gradu amaierako lan honen helburua Villapresenteko labirintotik irteteko ibilbide optimoa aurkitzea da. Hortaz, kontrolezko problema baten aurrean gaude. Hori dela eta, *Q-learning* metodoarekin lan egingo dugu problema horren ibilbide optimoa lortzeko.

Aipatzekoa da Villapresenteko labirintoa sari sakabanatuetako problema bezala defini daitekeela. Sari sakabanatuetako problemetan egoera guztiak 0 balioko saria dute, amaierako egoerako saria izan ezik. Villapresenteko labirintoaren kasuan, egoera guztiak -1 balioko saria dutenez, amaierakoa izan ezik, sari sabakanatuko problema moduan defini daiteke. Ezaugarri hauetako problema izateak asko zailtzen du ikasketa; alegia, Q-learning metodoari asko kostatzen zaio ikastea.

## 3.3 TD metodoak: Q-learning

Ezberdintasun tenporalezko metodoak (TD) baliogarriak dira eragilearen eta ingurunearen arteko interakzioetan oinarritzen diren problemak ebazteko. Gainera, bi ezaugarri nagusi dituzte. Alde batetik, esperientziatik zuzenean ikas dezakete inguruneko dinamika biltzen duen eredua ezagutu gabe. Beste aldetik, balioetsitako balioak eguneratu ahal dituzte ikasitako beste balioetsitako balioak oinarritzat hartuz.

Eragileak ez duenez ingurunea ezagutzen, ingurunearekin interakzionatu behar du, horri buruzko informazioa ikasteko eta bere portaera hobetzeko. Horretarako, erabilgarria izan ohi da urrats bakoitzean politika estokastikoa bat erabiltzea, eragileak egoera partikular batetatik akzio ezberdinak aukeratu ahal dituelako.

Bi ikaskuntza mota ezberdindu ahal dira eragileak ikasten ari duen politikan egiten duen erabileraren arabera: politika-on ikaskuntza eta politika-off ikaskuntza [3].

- **Politika-on ikaskuntza**: Eragileak $\pi$ politikaren bitartez ikasten du, $\pi$-k lortutako esperientziatik abiatuta. Hau da, erabakiak hartzeko erabilitako $\pi$ politika ebaluatzen eta hobetzen da.

- **Politika-off ikaskuntza**: Eragileak $\pi$ politikaren bitartez ikasten du, $\mu$ politikak lortutako esperientziatik abiatuta. Hau da, erabakiak hartzeko erabiltzen den politika, eta ebaluatzen eta hobetzen den politika ezberdinak eta independienteak dira haien artean.

Hala ere, gradu amaierako lan honetan Q-learning metodoa landuko dugunez, bakarrik **politika-off ikaskuntza** azalduko dugu.

**Off-politika ikaskuntza: Q-learning**

Off-politika ikaskuntzan bi politika desberdindu daitezke: portaera-politika eta estimazio-politika. Portera-politika eragileak jarraitzen duen politika da eta estimazio-politika, aldiz, ebaluatzen eta hobetzen den politika da. Esan bezala, bi politika horiek independienteak dira. Bestalde, estimazio-politika determinista izan ohi da eta portera-politika, ordea, estokastikoa.

Beraz, laburbilduz, off-politika ikaskuntzan, $V_\pi(s)$ edo $Q_\pi(s,a)$ kalkulatzeko $\pi$ politika ebaluatzen da $\mu$ politika jarraitzen den bitartean:

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu. \tag{3.1}$$

Beraz, behin off-politika ikaskuntzaren oinarria ezagututa, Q-learning metodoa aztertuko dugu. Hala ere, horretan sakondu baino lehen, bi estrategia mota definitu behar ditugu: *greedy* eta $\epsilon$-*greedy*.

### *Greedy* estrategia

Izenak dioen moduan, *Greedy*-k kodiziarekin lan egiten du. Ondorioz, eragileak $s$ egoeratik hasita esperotako itzulera handiena ematen dioen akzioa aukeratuko du. *Greedy* estrategia jarraitzen duen politika hurrengo eran definitzen da:

$$\pi'(s) = \underset{a \in \mathcal{A}}{argmax} Q(s, a). \tag{3.2}$$

Politika honekin, esperotako itzulerak ahalik eta gehien esplotatzen dira, baina akzioen portaerak gutxi esploratzen dira [4].

### $\epsilon$-*greedy* estrategia

Estrategia honekin, eragileak $\epsilon$ probabilitatearekin zoriz akzio bat aukeratuko du eta $1 - \epsilon$ probabilitatearekin, aldiz, akzio onena hautatuko du; hots, $s$ egoeratik abiatuta akzio-balio funtzio handiena ematen dioen akzioa. $m \in \mathbb{N}$ $s$ egoeratik abiatuta aukeratu ahal diren akzio posibleak izanik, $\epsilon$-*greedy* estrategia jarraitzen duen politika hurrengo eran definitzen da:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{m}, & a^* = \underset{a \in \mathcal{A}}{argmax} Q(s, a) \\ \frac{\epsilon}{m} & \text{beste kasutan} \end{cases} \tag{3.3}$$

Politika honekin, akzioen portaeren esplorazioa handitzen da, *greedy* politikaren bitartez aztertzen ez diren egoerak aztertu ahal direlako [4].

Bi estrategiak azalduta, Q-learning metodoa landuko dugu.

Lehen esan dugun bezala, Q-learning off-politika ikaskuntzaren metodo bat da. Beraz, portaera-politika eta estimazio-politika ezberdinak izango dira. Q-learning-en bitartez politika optimoa azkar lortzea zaila da, eragileak ez duelako inguruneari buruzko informazioa eta hori eskuratzeko ingurunea esploratu behar duelako urratsez urrats.

Teknika honek $Q(s, a)$ akzio-balioak erabiltzen ditu $s$ egoeratik $a$ akzioa aukeratuz, estimatzen den itzulera totalaren hurbilketa lortzeko. Hala ere, akzio-balio funtzio horiek eguneratzeko, aipatutako bi politikak hartzen ditu kontuan. Beraz, iterazioen bitartez, algoritmo honek hasierako $Q(s, a)$ akzio-balio funtziotik abiatuta, $Q_*(s, a)$ akzio-balio funtzio optimoa lortzen du, ondoren politika optimoa zehazteko. Ikus dezagun nola eguneratzen diren akzio-balio funtzioak.

Demagun $t$ denbora-urratsean egonda, $s$ egoeratik abiatuta $\mu$ portera-politika erabiliz hurrengo akzio bat aukeratzen dugula: $A_{t+1} \sim \mu(\cdot|S_t)$. Demagun, aldiz, $\pi$ estimazio-politika erabiliz hurrengo akzio alternatibo bat ere kontuan hartzen dugula: $A' \sim \pi(\cdot|S_t)$. Orduan, $t$ urratsean $s$ egoeratik hasita eta aukeratzen dugun benetako akzioaren $Q(S_t, A_t)$ akzio-balio funtzioa, akzio alternatiboaren baliora eguneratuko dugu. Hau da,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)). \tag{3.4}$$

Beraz, *boostrapping*[†] egiten dugu akzio alternatiboko akzio-balio funtziotik abiatuta, horren adierazten baitu zenbateko izango litzatekeen $Q(S_t, A_t)$ balioak eskuratutako saria eragileak estimazio-politika jarraitu izan balu. Hots, eguneraketa horren bitartez, $Q(S_t, A_t)$ akzio-balio funtzioa estimatzen dugu etorkizuneko estimazio bat kontuan hartuz.

Behin hori jakinda, esan beharra dago Q-learning algoritmoari buruz hitz egiten dugunean, $\pi$ estimazio-politika *Greedy* politikari egiten diola erreferentzia. Beraz, gorriz dagoen adierazpena sinplifika daiteke:

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \underset{a' \in \mathcal{A}(S_{t+1})}{argmax}\, Q(S_{t+1}, a')) \\ &= R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a') \end{aligned} \tag{3.5}$$

Hori dela eta, Q-learning metodorako lortzen dugun balioen eguneraketa $t$ urrats bakoitzerako hurrengo hau da:

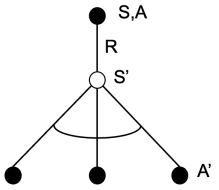
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a') - Q(S_t, A_t)), \tag{3.6}$$

non $\alpha \in [0, 1]$ parametroa ikasketa-faktorea den.

Aipatzeko da ikasketa metodoetan $\alpha$ parametroa episodioetan zehar txikitu behar dela. Alegia, $\alpha$-k gero eta gutxiago ikasi behar du.

Eguneraketa horren *backup*-diagrama hurrengoa izango da:

[†]**Boostrapping**: Zerbait estimatzen denean beste estimazio batean oinarrituz.




**3.2. irudia**: Q-learning-en *backup*-diagrama

Kontuan hartu behar da Q-learning-ek Bellman-en itxaropen ekuazio optimoa erabiltzen duela urratsez urrats balioak eguneratzeko. Beraz, edozein $s, a$ bikoterako $Q(s, a)$ akzio-balio funtzioak $Q_*(s, a)$ baliora konbergituko du. Horrela, politika optimoa lortuko da.

**Teorema 3.3.1.** *Q-learning algoritmoak akzio-balio funtzioara konbergitzen du,* $Q(s, a) \rightarrow Q_*(s, a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

Azkenik, esan behar dugu esplorazioa egiten diren metodoetan ezinbestekoa dela **GLIE** baldintza betetzea. GLIEk hurrengo adierazten du:

- Egoera-akzio bikote guztiak infinitu aldiz esploratu behar dira.
- Politikak *Greedy* politikara konbergituko du.

GLIE baldintza betetzeko aukera bat $\epsilon$-*greedy* politika erabiltzea da. Ondorioz, Q-learning metodoan portera-politika bezala $\epsilon$-*greedy* politika erabiltzea ohikoa da.

Atal honetan azaldutakoa kontuan hartuz, *Q-learning*-en algoritmoa ondorengoa izango da:

**Q-learning algoritmoa**

Hasieratu $Q(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ nahierara eta $Q(amaierakoegoera, \cdot) = 0$
**Errepikatu episodio (gertaera) bakoitzeko**
$S$ hasieratu
**Errepikatu episodioaren urrats bakoitzeko**
$S$-tik abiatuz $A$ aukeratu estimazio-politika erabiliz (adibidez: $\epsilon$-*greedy* politika)
$A$ akzioa aukeratu eta $R$,$S'$ behatu:
$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \max_{a \in \mathcal{A}(S')} Q(S', a) - Q(S, A)]$
$S \leftarrow S'$
**S amaierako egoera den arte**

# 4. kapitulua

# Kasu praktikoa: Villapresenteko labirintoa

Behin **balio-iterazioa** eta **Q-learning** algoritmoak azalduta, bi metodo horiek aplikatu ditugu gure probleman. Horretarako, *Python* programazio hizkuntza erabili dugu.

Lan honen helburua Villapresenteko labirintotik irtetzeko ibilbide eraginkorrena aurkitzea da. Horretarako, kontuan hartu behar dugu aipatutako bi algoritmoek ezaugarri ezberdinak dituztela; hots, balio-iterazioa problemaren informazio guztia ezagutzen denean erabil daiteke eta Q-learning, aldiz, eragileak ez duenean ingurunea ezagutzen.

## 4.1 Lehenengo urratsak

Lanean hasi baino lehen, labirintoaren mapa sakon landu behar izan dugu. Villapresenteko labirintoaren mapa B eraskinean dago eskuragarri, B.1. irudian hain zuzen ere.

Hala ere, mapa horren bitartez labirintoaren dimentsioak eta ezaugarriak lortzea zaila izan denez, mapa eskuz islatu da lauki-sare batean. Eskuz egindako Villapresenteko labirintoaren mapa B eraskinean dago B.2. irudian. Lauki zuriek posizio posibleak adierazten dituzte eta lauki beltzek, ordea, hormak islatzen dituzte. Labirintoa lauki-sarean irudikatuz problemaren hurrengo hiru ezaugarriak lortu ditugu.

Hasteko, ikusi dugu problema $71x71$ tamainako matrize baten bitartez adierazi ahal dela. Matrize horretan bi elementu ezberdin daude: 0 zenbakiak posizio posible bat adierazten du eta 1 zenbakiak, ordea, horma islatzen du. Bai *balio-iterazioa* eta bai *Q-learning* inplementatzeko, C eranskineko matrizea erabili dugu.

Bestalde, ikusi dugu problemak 2520 posizio ezberdin dituela, mapak 2520 lauki zuri eta 2521 lauki beltz baititu.

Azkenik, konturatu gara hasierako puntua $(38, 1)$ dela eta amaierakoa, aldiz, $(36, 1)$ puntua. Beraz, helburua da $(38, 1)$ puntutik $(36, 1)$ puntura ailegatzea ibilbide laburrena eginez, betiere kontuan hartuz soilik lauki zurietatik aurrera egin daitekeela.

Aipatutako lehenengo urrats horietan Villapresenteko labirintoaren informazio guztia lortu dugunez, lehenik eta behin problema MDP moduan definituko dugu.

## 4.2 Problema MDP moduan definituta

Problema MDP moduan definitzeko 1. kapituluan azaldu ditugun kontzeptuak kontuan hartu ditugu. Beraz, gure problema Markov-en erabaki-prozesu moduan definitzeko, $\omega =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ tupla osatzen duten parametroak zehaztu behar izan ditugu.

### 1 Egoerak

Problemak 2520 egoerak ditu eta egoera bakoitza labirintoaren lauki zuri batekin adierazita dago. Esan bezala, hasierako egoera $(38, 1)$ laukia da eta amaierako egoera, aldiz, $(36, 1)$ laukia.

### 2 Akzioak

Lau akzio posible egongo dira labirintoaren egoera batetik bestera mugitzeko: gora joan, behera joan, eskumara joan edo ezkerrera joan. Hala ere, kontuan hartu behar dugu egoera batzuetan lau akzioak ez direla posibleak.

### 3 Trantsizio-probabilitate funtzioa

Egoeren arteko trantsizio-probabilitate funtzioa definitzeko egoeren arteko trantsizio-matrizea erabili dugu. Matrize horrek egoera batetik, akzio bat hartuz, beste egoera batera mugitzeko probabilitatea adierazten du.

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{1,1} & . & . & . & \mathcal{P}_{1,2520} \\ . & . & . & . & . \\ . & . & . & . & \\ . & . & . & . & . \\ \mathcal{P}_{2520,1} & . & . & . & \mathcal{P}_{2520,2520} \end{bmatrix} \tag{4.1}$$

Trantsizio-matrizeekin lan egiteko, akzio bakoitzerako matrize bat sortu dugu.

Kontuan hartu behar dugu problema honetan transizio-matrizeak 0 eta 1 balioz osatuta dagoela: akzio zehatz batekin $i$ egoera batetik $j$ egoerara mugitzea posiblea bada $\mathcal{P}_{i,j} = 1$ balioa adieraziko da matrizean eta ez bada posiblea $\mathcal{P}_{i,j} = 0$ izango da. Bestalde, akzio horrekin $i$ egoeratik beste egoera batera mugitzea ezinezkoa bada, $\mathcal{P}_{i,i} = 1$ dela esango dugu.

### 4 Sari-funtzioa

Sari-funtzioa azaldu baino lehen, probleman zehar sariak nola banatu ditugun azaldu beharra dago. Helburua ibilbide optimoa aurkitzea denez, positiboki soilik saritu dugu amaierako egoera. Hortaz, amaierako egoerari +500 saria eman diogu eta gainontzeko egoera guztiei -1 balioa. Sariak modu horretan banatuta ditugunez, lehen esan dugun moduan, labirintoaren problema sari sakabanatuen problema dela esan ohi da.

Orain, aurreko elementuan bezala, sari-funtzioa, matrize baten bidez adieraz dezakegu akzioen arabera. Hala ere, ohartu egoera batean egonda, edozein akzioa eginez, irabaziko den saria berdina dela. Beraz, kasu honetan saria-funtzioa berdina izango da akzio guztietarako.

$$\mathcal{R} = \begin{bmatrix} \mathcal{R}_1 \\ . \\ . \\ \mathcal{R}_{2520} \end{bmatrix} \tag{4.2}$$

### 5 Deskontu-faktorea

$\gamma$ deskontu-faktorea 0 baliotik hurbil badago, berehalako sariei garrantzia handiagoa ematen zaie. Aldiz, 1 baliotik hurbil badago, etorkizuneko sariei ematen zaie garrantzia. Gure kasuan, problema ebazteko $\gamma = 0.999$ erabili dugu.

## 4.3 Balio-iterazioa: problemaren ebazpena eta emaitzak

Bigarren kapituluan esan dugun bezala, balio-iterazio metodoa erabili ahal izateko guztiz beharrezkoa da problemaren informazio guztia ezagutzea.

Aurreko atalean Villapresenteko labirintoaren problema Markov-en erabaki-prozesuaren bidez adierazi ahal izateko, suposatu dugu problemaren informazio guztia ezagutzen genuela. Beraz, hasierako egoeratik amaierako egoerara iristeko ibilbide optimoa lortu ahal izan dugu 4.2 atalean definitu ditugun elementuak balio-iterazio algoritmoan aplikatuz. Algoritmo horren inplementazioa D eraskinean dago.

Markov-en erabaki prozesuaren elementuak matrize moduan adieraz daiteke. Ondorioz, egoeraz egoera lan egin beharrean, matrizeekin eta bektoreekin lan egin dugu, ondorengo adierazpena erabiliz:

$$\begin{bmatrix} V_1 \\ . \\ . \\ . \\ V_{2520} \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ . \\ . \\ . \\ \mathcal{R}_{2520} \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{1,1} & . & . & . & \mathcal{P}_{1,2520} \\ . & . & . & . & . \\ . & . & . & . & \\ . & . & . & . & . \\ \mathcal{P}_{2520,1} & . & . & . & \mathcal{P}_{2520,2520} \end{bmatrix} \begin{bmatrix} V_1 \\ . \\ . \\ . \\ V_{2520} \end{bmatrix} \tag{4.3}$$

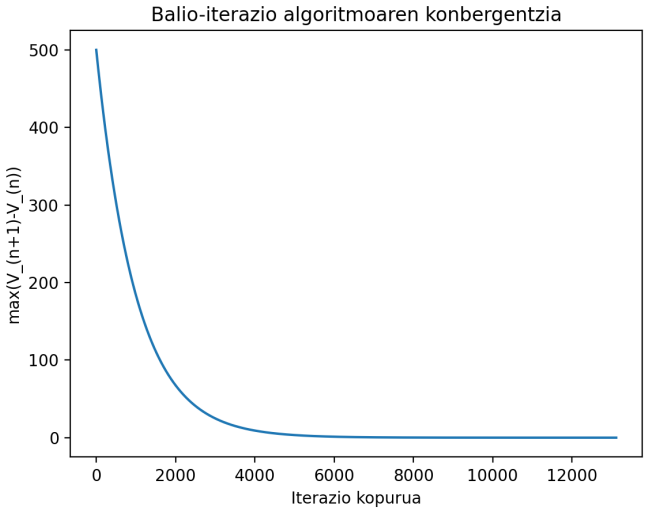
Balio-iterazio algoritmoaren lehenengo urratsa egoera bakoitzaren balio-funtzioa, $V(s)$, zoriz hasieratzea da. Balio horiek $V$ bektorean gorde ditugu. Ondoren, iterazio bakoitzean, akzio posible guztietarako, egoera bakoitzaren akzio-balio funtzioa kalkulatu dugu 4.3 adierazpena erabiliz. Villapresenteko labirintoan lau akzio posible definitu ditugunez, iterazio bakoitzean lau aldiz erabili dugu aurreko formula. Ohartu, lau kalkulu horietan aldatzen den elementu bakarra trantsizio-probabilitatea matrizea dela.

Iterazioan zehar lortutako akzio-balio funtzio guztiak hiztegi batean bildu ditugu. Hiztegi horrek labirintoaren $(s, a)$ posible guztien balioak gordetzen ditu.

Iterazio bakoitzaren amaieran, $s$ egoera bakoitzerako hiztegiko $(s,\ a_i)$ guztietatik balio maximoa gorde dugu. Horrela, $V$ bektorea eguneratu dugu eta bektore berri bat lortu dugu. Egoera bakoitzerako balio maximoa eskaintzen duen akzioak egoera horren politika zehaztu du.

Balio-iterazioak bukatuko da $n$. iterazioko eta $n + 1$. iterazioko egoera guztietarako balio-funtzioen kenketaren maximoa 0.001 baino txikiagoa denean. Hots, $V_{n+1}$ eta $V_n$ bektoreen arteko kenketa egiterakoan, balio guztietatik, maximoa 0.001 baino txikiagoa denean.

Beraz, balio-iterazioa erabilita, zailtasunik gabe Villapresenteko labirintoaren ibilibide optimoa lortu dugu. Algoritmo horrek 13000 iterazio inguru behar ditu helburua lortzeko.




**4.1. irudia**: Balio-iterazioaren konbergentzia

F eranskinean, F.1. irudian hain zuzen ere, ikus daiteke horiz margotuta zein den hasierako egoeratik amaierako egoerara ailegatzeko ibilbide optimoa. Gainera, berdez eta urdinez margoztutako egoerak ibilbide *alternatiboak* dira. Hau da, posizio horietatik pasatuz gero ere ibilbide optimoa lor daiteke. Izan ere, ibilbide optimo bat aurkitzeak ez du adierazi nahi ibilbide hori bakarra dela.

Gainera, F.1. irudian ikus daitekeen moduan, algoritmo horren bitartez labirintoaren edozein egoeraren politika optimoa lortzen dugu. Beraz, labirintoaren edozein egoeratik amaierako egoerara ailegatzeko ibilbide optimoa ere lortu dugu. Alegia, problemak guztiz konbergitu du soluzio optimora.

## 4.4 Problema errefortzuzko ikaskuntzaren bitartez ebazteko planteatuta

Behin frogatuta Villapresenteko labirintoaren informazio guztia ezagututa ibilbide optimoa lor dezakegula, orain beste baldintza desberdinetan aztertu dugu problema. Horretarako, suposatu dugu trantsizio-probabilitate funtzioa eta sari-funtzioa ezezagunak direla. Hortaz, bi elementu horiekin lan egin beharrean, eragilea eta ingurune osagaiekin lan egin dugu. Izan ere, suposatu dugu eragileak ez duela ingurunea ezagutzen. Hirugarren kapituluan esan den bezala, eragileak ingurunearekin interakzionatu behar du, horri buruzko informazioa ikasteko.

Baldintza horiek beteta, errefortzuzko ikaskuntzaren bidez ibilbide optimoa aurkitzen ahalengidu gara.

Metodo honetan, eragilea egoera batetik bestera mugitzen da aurreko atalean azaldu ditugun akzioak erabiliz (gora, behera, eskuma, ezkerra). Aukeratzen duen akzioaren arabera, inguruneak sari ezberdinak ematen dizkio eragileari. Era horretan, eragileak inguruneari buruzko informazioa jasotzen du. Ingurunearen sariak aurreko atalean azaldutakoak dira. Alegia, egileak amaierako egoerara ailegatzen bada +500 saria irabazten du, amaierara ailegatzea onena baita. Beste egoeretan, aldiz, -1 zigorra jasaten du, eragileak zenbat eta gehiago ibili orduan eta ibilbide luzeagoa egiten duelako.

Bestalde, eragileak inguruneari buruzko informazioa ikasteko $\epsilon$-*greedy* politika erabiltzen du. Problema honetarako, $\epsilon$ parametroari 0.4 balioa esleitu diogu. Beraz, 0.4 probabilitatearekin eragileak zoriz akzio bat aukeratzen du, eta 1-$\epsilon$= 0.6 probabilitatearen akzio onena hautatzen du.

Azkenik, $\gamma$ eta $\alpha$ parametroak finkatu ditugu. Hasteko, $\gamma$ parametroa, aurrekoan bezala, 0.999 ezarri dugu. Bestalde, $\alpha = \frac{1}{k(s,a)}$ finkatu dugu, non $k(s,a)$ balioak eragileak $s \in \mathcal{S}$ egoeran $a \in \mathcal{A}$ aukeratzea zenbatetan egin duen adierazten duen.

Behin eragileak inguruneari buruzko informazioa ikasteko erabiltzen dituen elementu guztiak definituta izanda, Q-learning metodoaren bitartez Villapresenteko labirintotik irtetzeko ibilbide optimoa aurkitzen saiatu gara.

## 4.5 Q-learning: problemaren ebazpena eta emaitzak

Aurreko ataleko elementuak kontuan hartuz, Q-learning algoritmoaren bitartez helburua lortzen saiatu gara. Hala ere, sari sakabanatuetako dimentsio handiko problema batean lan egiteak asko zaildu digu ibilbide optimoaren bilaketa. Izan ere, Q-learning algoritmoan etengabe egin dugu lan baldintza ezberdinetan. Azkenik, balio-iterazioarekin lortutako emaitza erabiliz, lortu dugu Q-learning-en bitartez hasierako egoeratik amaierako egoerara ailegatzea ibilbide motzena eginez. Hurrengo orrialdeetan azalduko dira egindako saiakuntza guztien baldintzak, ezaugarriak eta emaitzak.

Aipatzekoa da Villapresenteko labirintoarekin lan egin baino lehen, atergabe probak egin direla zoriz sortutako dimentsio txikiko problemekin.

Egindako saiakuntzetan denbora mugatu batean konbergentzia lortzeko

arazo asko izan ditugunez, bi ordenagailu ezberdinekin lan egitea erabaki dugu. Alde batetik, MacBook Air (prozesadorea: Intel Core i5 eta memoria RAM: 8 GB) ordenagailuarekin lan egin dugu, eta beste aldetik, HP Pavilion 15-b129es (prozesadorea: Intel Core i3 eta memoria RAM: 4GB) ordenagailuarekin.

Hasiera batean, ahalegindu gara 3.kapituluan adierazitako Q-learning algoritmo teorikoa inplementatzen eraldaketak egin gabe. Horretarako, lehenik eta behin, egoera eta akzio bikote posible bakoitzerako akzio-balio funtzioa zoriz hasieratu dugu. Balio horiek, balio-iterazioan azaldu dugun moduan, hiztegi baten bitartez bildu ditugu. Balioen bilketa hori **Q-taula** baten bitartez egin da.

Lanaren helburua (38,1) egoeraren eta (36,1) egoeraren arteko ibilbide optimoa bilatzea denez, episodio guztiak hasierako egoeratik hasieratzea erabaki dugu. Hortaz, baldintza horietan Q-learning algoritmoa inplementatu dugu. Hala ere, bi ordenagailuak 3 egunez lan egiten utzi ostean, ikusi dugu Q-learning-ek ez duela episodio bakarra ere burutzen. Hortaz, ondorioztatu dugu baldintza horietan eragilea ez dela gai amaierako egoera aurkitzeko.

Villapresenteko labirintoa hain handia denez, lehenengo episodioan eragileari ezinezkoa egin zaio amaierako egoera aurkitzea. Eragileak inguruneari buruzko informazioa lortzeko $\epsilon$-*greedy* politika erabiltzen duenez, gerta daiteke amaierako egoeratik hurbil egonda ere, ibilbidea kontrako norabidean egiten jarraitzea, eragileak oraindik ez dakielako amaierako egoera gailentzean inguruneak positiboki sarituko diola.

Horren ondorioz, algoritmoa beste baldintzetan inplementatzea erabaki dugu. Hasierako egoeratik amaierako egoerara igarotzeko ibilbidearen distantzia luzea denez, pentsatu dugu episodio guztiak zoriz aukeratutako egoeratik hasieratzea. Hau da, episodioaren hasieran labirintoaren edozein egoera zoriz aukeratu dugu, balitekeelako amaierako egoeratik hurbilago egotea. Hala ere, baldintza horietan Q-learning inplementatuz, konbergentzia denbora mugatuan lortzeko arazo izan ditugu. Aurreko kasuan bezala, 3 egun ibili gara bi ordenagailuekin lan egiten. Izan ere, oraingoan ere, eragileak ez du amaierako egoera aurkitu lehenengo episodioan.

Hortaz, konturatu gara hasierako episodioa amaierako egoeratik hurbilago aukeratuz gero, agian eragileari lana erraz diezaiokegula. Hortaz, hurrengo saiakeran *gradiente* moduko estrategia bat aplikatzea erabaki dugu, non hasierako episodioetan, egoera hasieratzeko, amaierako egoeratik hurbil dagoen zorizko egoera bat aukeratzen den. 50 episodio ondoren, episodioa hasieratzeko eremua pixka bat zabaldu da eta egoera eremu horretatik zoriz aukeratu da. Era horretan lan eginez, pixakana-pixkana eremua zabaldu

dugu, eta episodio asko egin ondoren, labirintoaren edozein egoeratik hasieratu daiteke episodioa. Hala eta guztiz ere, metodo honek bi alderdi negatibo ditu.

Alde batetik, nahiz eta labirintoaren hasierako eta amaierako egoeren arteko ibilbidea luze izan, bi posizio horiek hurbil daude haien artean. Beraz, ezaugarri horretako labirinto batean *gradientearen* teknika ezartzea oso zaila izan da. Alegia, teorikoki, hasierako egoera oso hurbil dago amaierako egoeratik, soilik landare-horma baten bitartez bereizita baitago. Horregatik, teknika horren bitartez, gerta daiteke hasierako episodioetan, egoera hasieratzeko, zoriz hasierako egoera aukeratzea. Horrek, lehenengo saiakeraren arazora eramaten gaitu. Beste aldetik, errefortzuzko ikaskuntzan teknika horiek erabiltzea ikaskuntzaren kontra joatea da, eragilea laguntzen baitugu amaierako egoeratik hurbil dagoen egoera batetik hasieratzen dugunean. Funtsean, nahiz eta baldintza horietan zenbait saiakuntzak egin, *gradientearen* teknika alde batera uztea erabaki dugu.

Hurrengo saiakeran episodio bakoitzaren urrats kopuruak mugatzea erabaki dugu. Era horretan, eragileak episodio batean etengabe esploratzea sahiestuko dugu. Episodio kopurua finkatzeko, balio-iterazioarekin lortutako politiken mapa kontuan hartzea komenigarria izan da. Horrela, ziurtatu dugu mugatutako urrats kopuruak egokiak direla; hots, ez direla ez gutxi ezta asko ere. Mapa aztertuz, ondorioztatu dugu ibilbide optimo luzeena 436 egoeretatik pasatzen dela. Datu hori kontuan hartuz, eragileak episodio bakoitzeko gehienez 800 urrats egitea erabaki dugu. Gainera, aurrekoan azaldutako guztia aztertu ondoren, episodio bakoitza egoera ezberdinetatik hasieratzea erabaki dugu (eta ez beti hasierako egoeratik).

Hurrengo pausoa episodio kopurua finkatzea izan da, errefortzuzko ikaskuntzaren bidez sari sakabanatuetako dimentsio handiko problemak soluzio optimoa noiz lortzen duen finkatzea oso zaila baita. Hasieran batean, saiatu gara problema osoaren soluzio optimoa aurkitzen. Hala ere, egoera guztien politika optimoa lortzeko denbora asko beharko genuke. Ondorioz, gure helburuan oinarrituta, saiatu gara algoritmoa noiz gelditu behar den zehazten, hasierako egoeratik amaierako egoerara ibilbide laburrena lortzeko.

Hasiera batean, saiatu gara **zikloekin** lan egiten. Ziklo bakoitzeko, eragileak 1000 episodio egiten ditu $\epsilon$-*greedy* politika jarraituz, eta horrez gain, 100 episodio egiten ditu *greedy* politikarekin. Azkenengo horrekin soilik ebaluatzen du ingurunea, eta, hori dela eta, Q-taularen balioak ez dira eguneratzen. Hortaz, ziklo bakoitzaren amaieran egindako 100 episodio horiek ziklo horretan egin den ingurunearen esplorazioa ebaluatzen dute.

Ziklo bakoitzerako ebaluazioaren **ratio** bat kalkulatu dugu:

$$ratioa = \frac{Amaierara.ailegatu}{100} \tag{4.4}$$

non *amaierara.ailegatu* 100 episodioetatik eragileak amaierara zenbat aldiz iritsi den adierazten duen. Probak egiten hasteko, Q-learning algoritmoa ratioa=1 berdintza 20 aldiz lortzean algoritmoa gelditzea ezarri dugu. Egindako saiakuntzetan ikusi dugu $\epsilon$ episodio bakoitzerako txikitzea eta $\alpha = \frac{1}{k(s,a)}$ finkatzea ez duela lagutzen konbergentzia denbora mugatuan lortzeko. Beraz, praktikoki, konbergentzia azkarrago lortzeko, hemendik aurrera beste parametro ezberdinekin lan egin dugu.

Alde batetik, $\epsilon$ episodio bakoitzerako txikitu beharrean, ziklo baten episodio bakoitzeko $\epsilon$ balioa txikitu dugu, eragileak ziklo horretan episodioak aurrera joan ahala ingurunea gehiago esplotatzeko. Gainera, $\epsilon = 0.05$ baino txikiagoa ez izatea baldintza ezarri dugu, beti esploratzeko aukera izateko. Bestalde, ziklo asko egiten dituenez, ziklo bakoitzaren hasieran eragileak $\epsilon = 0.4$ hasten da lan egiten. Beste aldetik, $\alpha = 0.1$ finkatu dugu, praktikoki parametro honekin emaitzak hobeagoak lortzen baititugu.

Estrategia berri hori oinarri moduan hartuta, bi ordenagailuekin hasi gara lanean probak egiten. Hala eta guztiz ere, erabaki dugu soilik ordenagailu batekin lan egitea, ziklo bat exekutatzeko denbora askoz handiagoa delako HP Pavilion ordenagailuan MacBook Air-en baino. Denboren analisia egiteko, ziklo bat 10 azpizikloetan banatu ditugu; hots, ziklo bat 100 episodiotan aztertu dugu. Alegia, hurrengoak dira bi ordenagailuek behar izan duten denbora lehenengo zikloa exekutatzeko*:

| 1. zikloa episodioetan | MacBook Air (s-tan) | HP Pavilion (s-tan) |
|---|---|---|
| 100 | 32.12 | 80.29 |
| 200 | 63.88 | 158.32 |
| 300 | 96.27 | 235.94 |
| 400 | 129.32 | 315.42 |
| 500 | 162.22 | 393.40 |
| 600 | 194.83 | 470.67 |
| 700 | 226.70 | 546.85 |
| 800 | 258.85 | 622.25 |
| 900 | 290.80 | 699.61 |
| 1000 | 322.56 | 774.30 |

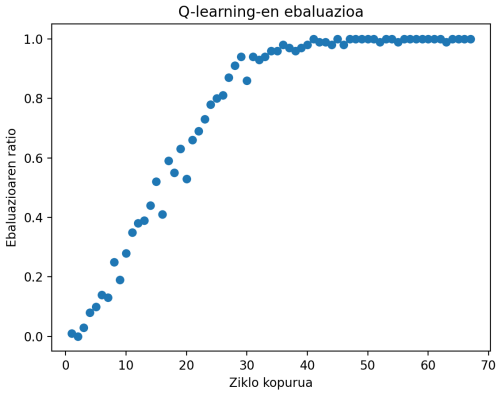
Zenbat eta ziklo gehiago egin, orduan eta denbora gutxiago behar izango dute ordenagailuek zikloa exekutatzeko, eragileak gero eta gehiago ezagutuko

*Denborak lortzeko ordenagailu bakoitzarako 5 aldiz exekutatu da 1.zikloa. Ondoren, lortutako emaitzak erabiliz bestekoak egin dira.

duelako ingurunea eta episodio gehiagotan ailegatuko delako amaierako egoerara. Hala ere, lehenengo zikloan bi ordenagailuek behar izandako denbora desberdina izan da. MacBookAir ordenagailuak 5 minutu eta erdi inguru behar izan ditu lehenengo zikloa egiteko, eta PH Pavilion-ak, aldiz, 13 minutu inguru. Arrazoi hori dela eta, MAC ordenagailuarekin lan egitea erabaki dugu.

MAC ordenagailua erabiliz, saiatu gara Q-learning algoritmoa inplementatzen ratioa=1 berdintzak 20 aldiz ematen duenerako.

4.2. grafikoan ikus daiteke nola zenbat eta ziklo gehiago egin, orduan eta handiagoa den ebaluazioaren ratio. Hori da esperotakoa, ziklo bakoitzean eragileak egiten dituen 1000 episodioetan inguruneri buruzko informazioa ikasten duelako eta egoera bakoitzeko akzio-balio funtzioak eguneratzen dituelako. Hori dela eta, ziklo bakoitzaren amaieran egiten den ebaluazioa gero eta hobeagoa izatea espero da, betiere kontuan hartuz ziklo batetik bestera ratioa ere gutxitu daitekeela.




**4.2. irudia**: Q-learning-en ebaluazioa (ratioa=1, 20 aldiz)

F eranskinean F.2. irudian ikus daitekeenez, ratioa=1 berdintza 20 aldiz ezarrita, hasierako egoeratik amaierako egoerara ibilbide optimoaren bidez iristea lortzen dugu. Baina, kontuan hartu behar dugu ratio hori ez dela adierazgarria ibilbidearen politika optimoa lortu dugun edo ez zehazteko. Estrategia horren bitartez bakarrik aztertzen dugu eragileak zikloetan zehar egiten duen lana hobera doan edo ez. Ratioa=1 berdintzak 20 aldiz betetzea berri onak dira eragilearen portaera hobetu delako adierazten duelako. Hala ere, nahiz eta baldintza horrekin exekutatutako kasu guztietan hasierako egoeratik amaierako egoerara doan ibilbide optimoa lortu (70.000 episodio inguru egiten dituelako), eragileak inguruneari buruzko informazioa

lortzeko $\epsilon$-*greedy* politika jarraitzen duenez, Q-learning exekutatzen dugun bakoitzean zoriaren esku dago estrategia horrekin helburua lortzea edo ez. Hau da, gerta daiteke baldintza horietan 100 proba egin ondoren, hasierako egoeraren eta amaierako egoeraren arteko ibilbide optimoa ez lortzea.

Hortaz, hurrengo saiakera egiteko, balio-iterazioan lortutako politiken mapa kontuan hartu dugu (F eranskineko F.1. irudia). Lehen esan dugun bezala, hasierako egoeratik amaierako egoerara iristeko ibilbide optimo bat baino gehiago daude. Hala ere, ibilbide optimo guztien artean soilik hartuko dugu kontuan horietako ibilbide optimo bat.

Balio-iterazioan lortutako politiken mapatik abiatuz, ibilbide optimo bat finkatu dugu. Hau da, ibilbide hori osatzen duten egoerak eta egoera bakoitzaren politika optimoa hiztegi batean gorde ditugu. Egoera bakoitza labirintoaren posizioarekin gorde dugu eta politika optimoa zenbaki baten bitartez adierazi dugu (1: "Gora joan", 2:"Eskumara joan", 3:"Behera joan" eta 4:"Ezkerrera joan"). Era horretan, dakigu zein den eragileak jarraitu behar duen portaera optimoa hasierako egoeratik amaierarako egoerara iristeko.

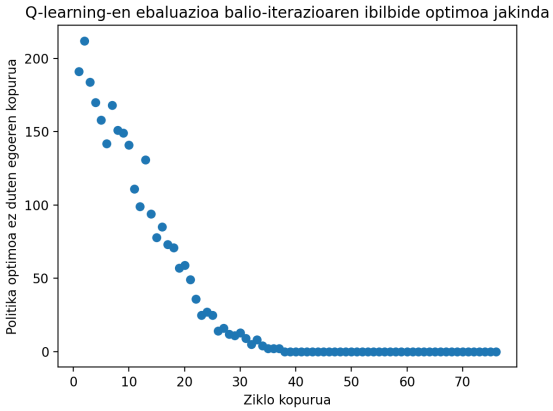
Beraz, ziklo bakoitzaren amaieran egin den ebaluaketan, ratioa kalkulatzeaz gain, gorde dugun ibilbide optimoarekin lan egin dugu. Ibilbide horrek zein egoeretatik pasatzen den dakigunez, ziklo bakoitzaren amaierara arte berrituta dugun Q-taulan egoera horiek aztertu ditugu. Egoera bakoitzaren politika finkatzeko, *greedy* estrategia erabili dugu; beraz, egoera bakoitzerako akzio-balio funtzio maximoak eskaintzen duen akzioa gorde dugu. Aurrean egin dugun bezala, akzio bakoitzari zenbaki bat esleitu diogu (1: "Gora joan", 2:"Eskumara joan", 3:"Behera joan" eta 4:"Ezkerrera joan"). Hortaz, ibilbide optimoaren egoera bakoitzaren politika eta ziklo horretara arte egoera bakoitzak esleituta duen politikak konparatu ditugu. Akzio bakoitzari zenbaki bat esleitu diogunez, egoera bakoitzaren politiken arteko kenketa egin dugu. Kenketan 0 lortzen badugu esan nahi du eragileak politika optimoa ikasi duela egoera horretarako, eta balio 0 ez bada, aldiz, ez. Kenketa egitearakoan egoera guztietan 0 balioa lortzen badugu, horrek adierazi nahi du eragileak politika optimoa ikasi duela.

Bestalde, saiakuntza horretan ratioa=1 berdintza 30 aldiz bete behar dela finkatu dugu. Horrela, lortutako politiken mapa aurreko saiakeran lortutakoarekin konpara dezakegu, ziurtatzeko zenbat eta episodio gehiago egin orduan eta hobeagoa izango dela eragilearen portarea. Algoritmo horren inplementazioa, estrategia hau erabiliz, E eranskinean dago.

Hortaz, laburbilduz, Q-learning algortimoaren exekuzioa gelditzeko bi baldintzak ezarri ditugu: behintzat ibilbide optimo bat lortzea hasierako

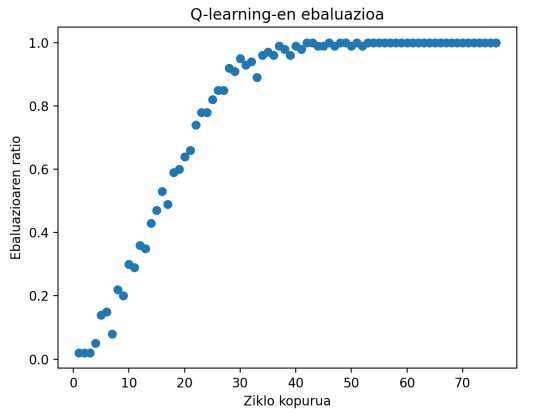
eta amaierako egoeren artean, eta ratioa=1 berdintza 30 aldiz lortzea.

Hurrengo grafikoetan ikus daiteke egindako ebaluaketa. Alde batetik, 4.3. grafikoak islatzen du nola zikloetan zehar eragileak ibilbide optimoa osatzen duen egoera bakoitzaren politika optimoa ikasten duen.




**4.3. irudia**: Q-learning-en ebaluazioa (balio-iterazioan lortutako ibilbidea kontuan hartuz)

Beste aldetik, 4.4. grafikoak adierazten du eragileak bere portaera hobetzen doala episodioetan zehar. Gogoratu azkenengo hori ez dela politika optimoaren adierazlea, bakarrik erabiltzen dugu, denbora aurrera egina ahala, eragileak bere portaera hobetzen duela ikusteko.




**4.4. irudia**: Q-learning-en ebaluazioa (ratioa=1, 30 aldiz)

F eranskinean F.3. irudian ikus daitekeenez, lanaren helburua lortu dugu. Hots, eragilea hasierako egoeratik amaierako egoerara iritsi da ibilbide optimo baten bitartez. Funtsean, estrategia horren bitartez, ziurtatzen dugu beti lortuko dugula bi posizio horien arteko ibilbide optimoa.

Gainera, balio-iterazioaren bidez F eranskinean dagoen F.1. irudiaren laguntzaz, estretegia honen bitartez lortutako emaitza F eranskinean dagoen F.2. politiken maparekin konparatu ahal izan dugu. Alderaketa horretan, ondorioztatu dugu, episodio gehiago eginez, espero zitekeen moduan, problema osatzen duten egoerek politika optimora konbergituko dutela.

## 4.6 Ondorioak

Lehenik eta behin, azpimarratu behar dugu lanaren helburu nagusia lortu dugula. Hots, Villapresenteko labirintoaren ibilbide optimoa lortu dugu errefortzuzko ikaskuntzaren bitartez. Hori lortzeko, errefortzuzko ikaskuntzaren oinarria den programazio dinamikoa ulertzea beharrezkoa izan da. Programazio dinamikoaren balio-iterazio algoritmoaren bitartez lortutako emaitzatik abiatuz, azkenean, gure helburua lortu dugu Q-learning algoritmoa erabiliz. Gainera, Q-learning-en exekuzioa gelditzeko balio-iterazioaren emaitza kontuan hartuta, baiezta dezakegu ibilbide optimo bat, behintzat, beti lortuko dugula. Hala ere, aipatu beharra dago Q-learning metodoaren bitartez helburua lortzeko zailtasun asko izan ditugula.

Bestalde, balio-iterazioa eta Q-learning-en eraginkortasunari buruzko ondorioak ere azpimarratzea garrantzitsua da. Aipatutako bi algoritmoak ezin dira elkar artean konparatu, ezaugarri ezberdinak dituztelako. Alde batetik, lanean zehar islatu den bezala, balio-iterazio algoritmoa problemaren informazioa guztia ezagutzen denean bakarrik erabil daiteke. Q-learning algoritmoa, aldiz, problemaren informazio guztia ezagutzen ez denean aplikatzea posiblea da. Beste aldetik, aipatutako ezaugarrien ondorioz, balio-iterazioak iterazio batean egoera guztien balio-funtzioa eguneratzen duen bitartean, Q-learning-ek egoera eta akzioen bikoteen balioak baino ez ditu eguneratzen eragileak episodio bakoitzean egiten duen ibilbidearen arabera.

Villapresenteko labirintoa dimentsio handikoa denez eta sariak sakabanatuak dituenez, Q-learning-en algoritmoaren bitartez, bereziki, hasierako eta amaierako egoeren arteko ibilbide optimoa lortzea zaila da. Ondorioz, funtsezkoa da estrategia bat ezartzea Q-learning-en exekuzioa noiz gelditu behar den finkatzeko. Ikusi dugun bezala, episodio bakoitzaren urrats kopurua zehazten ez bada, denbora mugatu batean ibilbide optimoa lortzea oso zaila da. Eragilearentzat ingurunea ezezaguna da. Informazio

gabezia horren ondorioz, lehenengo episodioan egiten duen ibilbide zoriz egiten du. Izan ere, lehenengo episodioan amaierako egoerara iristekotan, ibilbidearen azken aurreko egoera baino ez da eguneratuko +500 saria kontuan hartuz. Ondorioz, eragileak zorizko ibilbidea egingo du bigarren zein lehenengo episodioan,

Gainera, sari sakabanatuen dimentsio handiko problemetan ebaluaketa prozesu horretan eragileak egiten duen lana ebaluatzea komenigarria da. Horrela ikusiko dugu nola episodioak aurrera egin ahala, eragileak bere politika hobetzen duela. Dena den, betiere kontuan hartu behar dugu irizpide horiek ez direla baliogarriak politika optimoa aurkitzeko.

Balio-iterazio metodoak, aldiz, Q-learning-ekin alderatuta soluzio optimora azkar konbergitzen du. Alegia, 13000 episodio inguru behar ditu labirintoaren politika optimoen mapa egiteko. Algoritmo horren eraginkortasuna aprobetxatuz, Q-learning-en bidez hasierako egoeraren eta amaierako egoeraren arteko ibilbide optimoa lortu dugu.

Hortaz, behin guztia aztertuta, esan dezakegu sari sakabanatutako dimentsio handiko problemak errefortzuzko ikaskuntzaren bitartez ebaztea oso zaila dela. Bi egoeren arteko ibilbide optimoa aurkitzea zaila bada, labirinto guztiaren politika optimoa lortzea ia ezinezko da, denbora asko beharko litzatekeelako.

Erabilitako ekipo informatikoen potentzia ere garrantzitsua izan da problemaren ebazpenean. Izan ere, egiaztatu da exekuzio-denbora luzea behar dituzten algoritmoak ebazteko ezinbestekoa dela prestazio onak dituen ordenagailu bat edukitzea (ram memoria zabala eta prozesadore indartsua), aipatutako exekuzio-denbora nabarmen murrizten baita.

Amaitzeko, ondorio gisa gustatuko litzaidake aipatzea gaiari buruz ahalik eta informazio gehien eskuratzeko ahalegin guztiak egin ditugun arren, proiektua iraupen mugatukoa izateak errefortzuzko ikaskuntzaren beste teknika batzuk ezartzea edo garatzea galarazi egin duela. Aplikatu ezin izan diren tekniken artean balio-funtzioaren hurbilketa azpimarra daiteke, besteak beste. Metodo hori egokia da hainbat egoera dituen problema ebazteko, soluzio optimora azkarrago konbergitzen du eta. Erabilgarria da ere sare neuronalekin edo adimen artifizialaren zerikusia duten anbizio handiko helburuak lortzeko.

Nork daki, ordea, matematikari gisa etorkizunean teknika horiek aplikatzeko aukerarik izango ez ote dudan?

# A. eranskina

# Frogak

**Teorema A.0.1.** *Edozein Markov-en erabaki-prozesurako:*

- *Existituko da beste edozein politiken baino edo bezain hobea den $\pi_*$ politika optimoa, $\pi_* \geq \pi \ \forall \pi$.*
- *Politika optimo guztiek egoera-balio funtzio optimoa lortzen dute: $V_{\pi_*}(s) = V_*(s) \forall s \in \mathcal{S}$.*
- *Politika optimo guztiek akzio-balio funtzio optimoa lortzen dute: $Q_{\pi_*}(s,a) = Q_*(s,a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.*

*Froga.* Teorema frogatzen hasi baino lehen, lema sinple bat frogatuko dugu.

**Lema A.0.2.** *Edozein bi politika optimoetarako, $\pi_1$ eta $\pi_2$, $s \in \mathcal{S}$ guztietarako $V_{\pi_1} = V_{\pi_2}$ berdintza betetzen da.*

*Froga.* Enuntziatuaren arabera, $\pi_1$ politika optimoa da. Beraz, politika optimoaren definizioaren arabera, $V_{\pi_1}(s) \geq V_{\pi_2}(s)$ da $s \in \mathcal{S}$ guztietarako.

Era berean, enuntziatuaren arabera, $\pi_2$ politika optimoa da. Beraz, politika optimoaren definizioaren arabera, $V_{\pi_1}(s) \leq V_{\pi_2}(s)$ da $s \in \mathcal{S}$ guztietarako.

Horrek inplikatzen du $V_{\pi_1} = V_{\pi_2}$ berdintza betetzen dela $s \in \mathcal{S}$ guztietarako. □

Lemaren ondorioz, teorema frogatzeko egin behar den bakarra da $\pi_*$ politika optimoa ezartzea, egoera-balio funtzio optimoa eta akzio-balio funtzio optimoa lortzen duena. Horretarako, hurrengo politika determinista (politika optimoa izateko hautagaia) definituko da, $\pi : \mathcal{S} \to \mathcal{A}$:

$$\pi_*(s) = \underset{a \in \mathcal{A}}{argmax} Q_*(s,a), \forall s \in \mathcal{S}. \tag{A.1}$$

Hasteko frogatuko da $\pi_*$-k egoera-balio funtzio optimoa lortzen duela. Alde batetik, $\pi_*(s) = \underset{a \in \mathcal{A}}{argmax} Q_*(s,a) \forall s \in \mathcal{S}$ denez, eta beste aldetik, $V_*(s) =$

$\underset{a \in \mathcal{A}}{max} Q_*(s,a), \forall s \in \mathcal{S}$ berdintza betetzen denez, $\pi_*$-k erabakitzen du egoera bakoitzeko akzio optimoa. Honek $V_*$ egoera-akzio funtzioa ematen du. Beraz, egoera bakoitzean $\pi_*$ politika optimoa jarraitzeak eragiten du egoera-funtzio optimoa eta egoera-funtzio optimoa berdinak izatea. Hots, $V_{\pi_*}(s) = V_*(s)$.

Era berean, froga daiteke $Q_{\pi_*}(s,a) = Q_*(s,a)$ berdintza betetzen dela $s \in \mathcal{S}$ eta $a \in \mathcal{A}$ guztietarako.

Azkenik, frogatuko dugu $\pi_*$ politika optimoa dela. Absurdura eramanez, demagun $\pi_*$ ez dela politika optimoa. Orduan, existitzen da $\pi$ politika bat eta $s \in \mathcal{S}$ egoera bat non, $V_\pi(s) > V_{\pi_*}(s)$ den. Gainera, frogatu dugu $V_{\pi_*}(s) = V_*(s)$ betetzen dela, hortaz, $V_\pi(s) > V_*(s)$. Baina hori ez da posible, definizioz $V_*(s) = \underset{a \in \mathcal{A}}{max} V_\pi$ delako. □

**Teorema A.0.3.** *Balio-iterazioak egoera-balio funtzio optimora konbergitzen du,* $V(s) \to V_*(s)\ \forall s \in \mathcal{S}$.

*Froga.* Edozein $\hat{V}$ balio-funtzioaren balioespenerako, Bellman *backup* eragilea definituko dugu.

Izan bedi $B : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$, non

$$B\hat{V}(s) = \max_{a\in\mathcal{A}}(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \hat{V}(s')) \tag{A.2}$$

den.

Hasteko, ikusiko dugu edozein $V_1, V_2$ balio-funtzioen balioespenerako Bellman eragilea *kontrakzio* bat dela. Horretarako, ikusi behar dugu hurrengo inekuazioa egia dela:

$$\max_{s\in\mathcal{S}} |BV_1(s) - BV_2(s)| \leq \max_{s\in\mathcal{S}} |V_1(s) - V_2(s)| \tag{A.3}$$

*Froga.*

$$\begin{aligned}
\max_{s\in\mathcal{S}} |BV_1(s) - BV_2(s)| = & \\
&= \max_{s\in\mathcal{S}} |\max_{a\in\mathcal{A}}(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_1(s')) - \\
&\max_{a\in\mathcal{A}}(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_2(s'))| \\
&= \max_{s\in\mathcal{S}} \gamma |\max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_1(s') - \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_2(s')| \\
&\leq \max_{s\in\mathcal{S}} \gamma \max_{a\in\mathcal{A}} |\sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_1(s') - \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a V_2(s')| \\
&= \max_{s\in\mathcal{S}} \gamma \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a |V_1(s') - V_2(s')| \\
&\leq \gamma \max_{s\in\mathcal{S}} |V_1(s) - V_2(s)|
\end{aligned} \tag{A.4}$$

Ohar moduan, esan beharra dago laugarren lerroaren desberdintza betetzen dela hurrengo propietatearengatik:

$$|\max_x f(x)| - |\max_x g(x)| \leq \max_x |f(x) - g(x)| \tag{A.5}$$

eta azkenenengo lerroaren desberdintza betetzen dela $\mathcal{P}_{ss'}^a$ positiboak direlako eta batuketa bat balioaren berdina delako. □

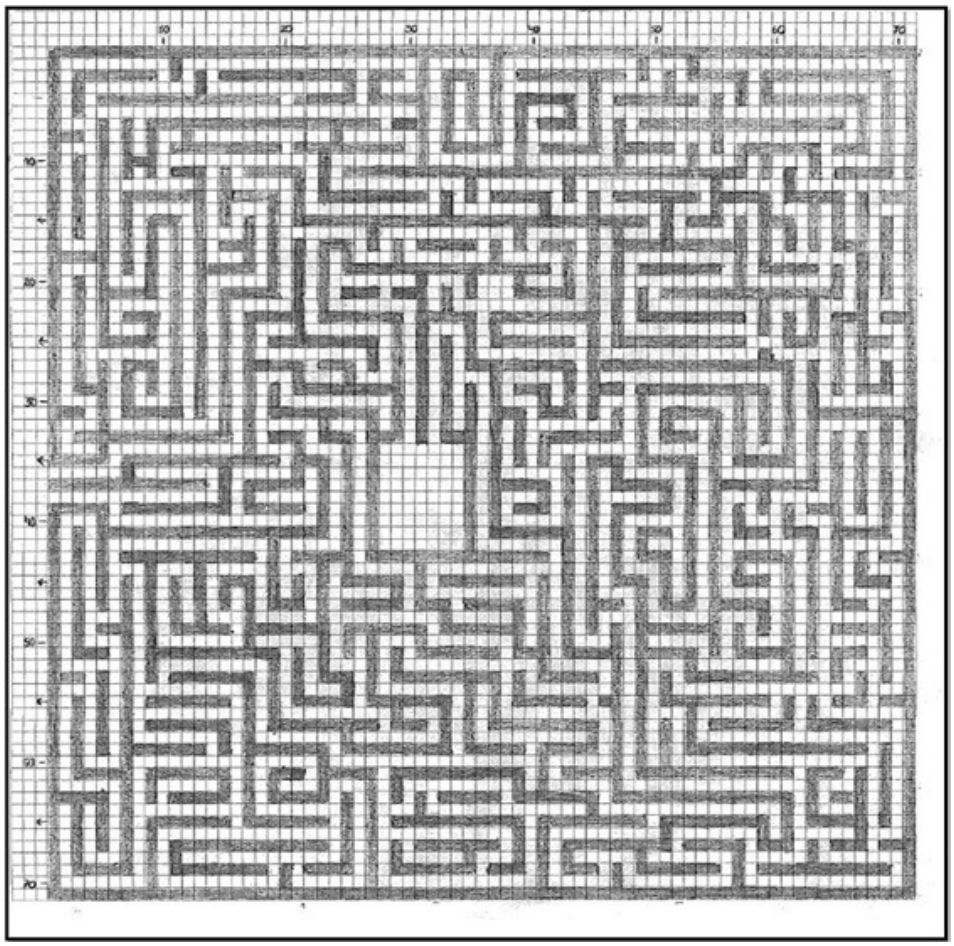
Frogatu dugu Bellman eragilea *kontrakzio* bat dela.

Beraz, $\hat{V}$ bektore batetik hasita, behin eta berriz iteratuz, nahitaez $V^*$ bektore batera konbergituko du, kontrakzioaren propietateak baieztatzen baitu puntu finkoaren existentzia eta bakartasuna. □

# B. eranskina

# Villapresenteko labirintoaren irudiak

**B.1. irudia**: Villapresenteko labirintoaren mapa



**B.2. irudia**: Villapresenteko labirintoaren mapa eskuz

# C. eranskina

# Villapresenteko labirintoaren kodifikazioa

Hau da Villapresenteko labirintoa kodifikatua 0 eta 1 zenbakiekin:

---

```
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1],[1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1],[1,0,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,
1,1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1,
1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1],[1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,
0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1],[1,0,1,0,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,
1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1],[1,0,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,
1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,
0,1],[1,0,0,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,
1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,0,1,0,1,0,1],[1,1,1,0,1,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,1,0,1,0,1],[1,0,0,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,
1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,1,1,1,
0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1],[1,0,1,0,1,0,1,1,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,1,0,0,0,0,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,1],[1,0,1,0,1,0,
1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,0,1],
[1,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,
```

```
0,0,0,0,0,1],[1,0,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,
1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,
0,1,1,1,1,1,1,0,1,0,1,1,1],[1,0,1,0,1,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,
1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,
0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1],[1,0,1,0,1,0,1,0,1,1,1,1,0,1,0,
1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,0,1],[1,0,1,0,1,0,1,0,
1,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1],[1,0,
0,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,1,1,
0,1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,0,
1,0,1],[1,1,1,1,1,0,1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,1,
0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,
0,0,0,0,1,0,1,0,1],[1,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,0,
1,0,1,1,1,1,1,0,1,0,1,0,1,0,1],[1,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,0,
0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,1,0,1],[1,0,1,0,1,1,1,1,1,1,0,1,
0,1,0,1,1,1,1,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,0,0,1,1,1,1,1,1,0,
1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1],
[1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,
1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,
1,0,1,1,1],[1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,
0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,
1,0,1,0,1,0,1,0,0,0,1],[1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,
1,1,1,0,0,0,1,1,1,0,1,0,1,1,1,0,1],[1,0,1,0,0,0,0,0,0,0,1,0,1,0,1,
0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1],[1,0,1,0,1,0,1,1,1,
0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,
1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1],[1,0,1,
0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,
1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,
0,1],[1,0,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,0,
1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,
0,1,0,1,1,1,0,1],[1,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,
0,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,1,
0,1,0,1,0,1,0,0,0,0,0,0,0,1],[1,1,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,0,
1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,0,1,
1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1],[1,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,1,0,1,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,
0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,1,1,
```

```
1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,
0,1,1,1,1,1,1,1,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1],[
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,
0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,
1,0,1,0,1],[1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1,
0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,
1,0,1,0,1,0,1,0,1,0,1],[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,
0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,
1,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1],[1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,
0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,
1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1],[0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,
1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,1,0,1,0,0,1],[1,1,1,
1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,
1,0,1,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,
0,1],[1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,
0,0,0,0,1,0,1,0,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,
0,1,0,0,0,1,0,1],[1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,
1,0,1,0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,
1,1,0,1,1,1,0,1,1,1,1,1,1,0,1],[1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,0,
0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1],[1,0,1,0,1,0,1,1,1,1,1,1,1,
1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,
0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1],[1,0,1,0,1,0,
0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
0,0,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1],[
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,
1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,0,
1,1,1,0,1],[1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,0,
0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,1,0,0,
1,0,1,0,1,0,0,0,0,0,1],[1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,
0,1,0,1,1,1,1,1,1,0,1,1,1,0,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,1,0,
1,1,1,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1],[1,0,1,0,0,0,1,0,1,0,1,0,1,0,0,
0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,1,0,1,0,0,0,
1,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,1],[1,0,1,0,1,1,1,0,1,
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,0,
1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,0,1],[1,0,1,
0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,
0,0,1,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,1,
0,1],[1,0,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,
1,1,1,1,1,1,1,0,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1,
0,1,1,1,0,1,0,1],[1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,
0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,
0,0,0,1,0,0,0,1,0,0,0,1,0,1],[1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,0,
```

```
1,0,1,0,1,1,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,
0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1],[1,0,0,0,1,0,1,0,0,0,0,0,
0,0,0,0,1,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,
0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1],[1,0,1,0,1,0,
1,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,
1,1,1,1,1,1,0,1,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,0,1,1,1],[
1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,
0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,
0,0,0,0,1],[1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,1,
0,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,
1,0,1,1,1,1,1,1,1,1,0,1],[1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,
0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],[1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,0,1,
0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,0,1,0,
1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1],[1,0,1,0,0,0,1,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,
1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,
1,1,1,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1,
0,1],[1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,
0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,1,0,1,0,1],[1,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,
1,0,1,0,1,0,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,0,1,0,1,1,1,0,1,0,1],[1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,1],[1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,
1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,1],[
1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,1,1,
1,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,0,1,0,1,1,
1,0,1,1,1],[1,0,1,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,
0,1,0,0,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,
0,0,1,0,0,0,0,0,0,0,1],[1,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,
1,1,1,1,0,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,1,1,1,1,
1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1],[1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],[1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
```

# D. eranskina

# Balio-iterazio algoritmoa

Hurrengo orrialdeetan *Pyhton*-en bitartez egindako balio-iterazioaren algoritmoa dago. Bai algoritmoan erabilitako funtzio osagarriak daude eta bai algoritmoa bera. Funtzio hauen bitartez, balio-iterazioa erabiliz, Villapresenteko labirintoaren ibilbide optimoa lortu dugu.

```
import math
import numpy
import random
import numpy as np
from pylab import *
import matplotlib.pyplot as plt

# Labirintoaren egoera/akzio bikote guztiak hiztegi batean bilduta.
# Q(s,a)=0 hasieran (nola hasieratu balio hauek berdin dio kasu honetan)

#(egoera,akzioa) posiblea bada mug=0 eta ez bada posiblea mug=1.
def egoeraposiblea(egoera, labirintoa, ak): #Egoera= (i,j)
    i=egoera[0] # Lehen elementua
    j=egoera[1] # Bigarren elementua
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    mug=0
    if ak=="Gorantz":
        if i-1<0 or labirintoa[i-1][j]==1:
            mug=1

    elif ak=="Beherantz":
        if i+1> (zabalera-1)  or labirintoa[i+1][j]==1:
            mug=1

    elif ak=="Eskumara":
```

```
        if  j+1> (luze-1) or labirintoa[i][j+1]==1:
            mug=1

    elif ak=="Ezkerrerantz":
        if j-1<0 or labirintoa[i][j-1]==1:
            mug=1

    return mug

#Hiztegia Q(s,a) guztiekin
def informazioaosatu(labirintoa):
    infor={}
    for i in range(len(labirintoa)):
        for j in range(len(labirintoa[0])):
            if labirintoa[i][j]!=1:
                for k in [1,2,3,4]:
                    if k==1:
                        ak="Gorantz"
                        if egoeraposiblea((i,j), labirintoa, ak)==0:
                            infor[(i,j), ak]=0

                    if k==2:
                        ak="Beherantz"
                        if egoeraposiblea((i,j), labirintoa, ak)== 0:
                            infor[(i,j), ak]=0
                    if k==3:
                        ak="Eskumara"
                        if egoeraposiblea((i,j), labirintoa, ak)== 0:
                            infor[(i,j), ak]=0
                    if k==4:
                        ak="Ezkerrerantz"
                        if egoeraposiblea((i,j), labirintoa, ak)== 0:
                            infor[(i,j), ak]=0

    return infor


#EGOERAK
#Egoera guztiak biltzen dituen lista. 1. errenkadako zutabe guztiak
#zehazkatzen ditu, ondoren 2. errenkadako zutabe guztiak, etab.
def egoeralista(labirintoa):
    egoeralis=[]
    zabalera= len(labirintoa) #arriba a abajo (NUM FILAS)
    luze= len(labirintoa[0]) #izquierda a derecha (NUM COLUM)
```

```
    kont=0
    for i in range(zabalera):
        for j in range(luze):
            if labirintoa[i][j]==0:
                egoeralis.append((i,j))

    return egoeralis

# Egoera bakoitzari zenbaki bat esleitzeko.
def egoeraguztiak(labirintoa):
    egoerak={}
    zabalera= len(labirintoa) #arriba a abajo (NUM FILAS)
    luze= len(labirintoa[0]) #izquierda a derecha (NUM COLUM)
    kont=0
    for i in range(zabalera):
        for j in range(luze):
            if labirintoa[i][j]==0:
                egoerak[(i,j)]= kont
                kont=kont+1

    return egoerak


## TRANTSIZIO—PROBABILITATE MATRIZEA (akzio bakoitzerako)

#akzioa=Gora
def MatrizeValItGora(labirintoa,egoerak):
    amaierakoegoera= (35,0)
    dimentsioa=len(egoerak)
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])

    PmatrizeaGora=np.zeros((dimentsioa,dimentsioa))

    for i in range(zabalera):
        for j in range(luze):
            if (i,j) in egoerak and (i,j)!=amaierakoegoera:

                if i>0 and labirintoa[i—1][j]!=1:

                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i—1,j)]
                    PmatrizeaGora[Pfila][Pcolumna]=1
```

```
                else:

                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j)]
                    PmatrizeaGora[Pfila][Pcolumna]=1

            elif (i,j) in egoerak and (i,j)==amaierakoegoera:
                Pfila=egoerak[(i,j)]
                Pcolumna=egoerak[(i,j)]
                PmatrizeaGora[Pfila][Pcolumna]=1

    return PmatrizeaGora

#akzioa= behera
def MatrizeValItBehera(labirintoa,egoerak):
    amaierakoegoera= (35,0)
    dimentsioa=len(egoerak)
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])

    PmatrizeaBehera=np.zeros((dimentsioa,dimentsioa))

    for i in range(zabalera):
        for j in range(luze):
            if (i,j) in egoerak and (i,j)!=amaierakoegoera:
                if i<(zabalera-1) and labirintoa[i+1][j]!=1:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i+1,j)]
                    PmatrizeaBehera[Pfila][Pcolumna]=1

                else:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j)]
                    PmatrizeaBehera[Pfila][Pcolumna]=1


            elif (i,j) in egoerak and (i,j)==amaierakoegoera:
                Pfila=egoerak[(i,j)]
                Pcolumna=egoerak[(i,j)]
                PmatrizeaBehera[Pfila][Pcolumna]=1

    return PmatrizeaBehera

#akzioa=eskuina
```

```
def MatrizeValItEskuina(labirintoa,egoerak):
    amaierakoegoera=(35,0)
    dimentsioa=len(egoerak)
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])

    PmatrizeaEskuina=np.zeros((dimentsioa,dimentsioa))

    for i in range(zabalera):
        for j in range(luze):
            if (i,j) in egoerak and (i,j)!=amaierakoegoera:
                if j<(luze-1) and labirintoa[i][j+1]!=1:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j+1)]
                    PmatrizeaEskuina[Pfila][Pcolumna]=1
                else:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j)]
                    PmatrizeaEskuina[Pfila][Pcolumna]=1


            elif (i,j) in egoerak and (i,j)==amaierakoegoera:
                Pfila=egoerak[(i,j)]
                Pcolumna=egoerak[(i,j)]
                PmatrizeaEskuina[Pfila][Pcolumna]=1

    return PmatrizeaEskuina

#akzioa=ezkerra
def MatrizeValItEzkerra(labirintoa,egoerak):
    amaierakoegoera=(35,0)
    dimentsioa=len(egoerak)
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])

    PmatrizeaEzkerra=np.zeros((dimentsioa,dimentsioa))

    for i in range(zabalera):
        for j in range(luze):
            if (i,j) in egoerak and (i,j)!=amaierakoegoera:
                if j>0 and labirintoa[i][j-1]!=1:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j-1)]
                    PmatrizeaEzkerra[Pfila][Pcolumna]=1
```

```
                else:
                    Pfila=egoerak[(i,j)]
                    Pcolumna=egoerak[(i,j)]
                    PmatrizeaEzkerra[Pfila][Pcolumna]=1

            elif (i,j) in egoerak and (i,j)==amaierakoegoera:
                Pfila=egoerak[(i,j)]
                Pcolumna=egoerak[(i,j)]
                PmatrizeaEzkerra[Pfila][Pcolumna]=1

    return PmatrizeaEzkerra

## SARIAK
#Labirintoaren matrizeen saria. Amaierako egoeraren saria +500
#izango da eta beste egoera guztien saria —1.
def matrizesaria(labirintoa):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    sariak=numpy.zeros(shape=(zabalera,luze))
    posfinala=(35,0)
    for i in range(len(labirintoa)):
        for j in range(len(labirintoa[0])):
            if labirintoa[i][j]==0 and (i,j)!=posfinala:
                sariak[i][j]=—1
            elif labirintoa[i][j]==0 and (i,j)==posfinala:
                sariak[i][j]=500
            else:
                sariak[i][j]=None
    return sariak

#Labirintoaren sariak array moduan ondoren Balio—iterazio algoritmoan
#erabiltzeko.

def sariakVI(labirintoa):
    egoerak=egoeraguztiak(labirintoa)
    amaierakoegoera=(35,0)
    dif=egoerak[amaierakoegoera]
    dimentsioa=len(egoerak)
    M=np.zeros((dimentsioa,1)) —1
    M[dif,0]=500

    return M

## Matrize huts honen bitartez politikak irudikatuko ditugu hurrengo ditugu
```

```
#hurrengo funtzioaren bitartez.
def matrizhutsa(labirintoa):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    M=[]
    for i in range(zabalera):
        m=[]
        for j in range(luze):
            m.append(0)
        M.append(m)
    return M

## Funtzio honek episodio bakoitzeko V(s) eta egoera bakoitzaren
#politika itzultzen du

# Eskuma—Ezkerra—Gora—Behera "\u2725"
# Eskuma "\u2192"
# Ezkerra "\u2190"
# Gora "\u2191"
# Behera "\u2193"
# Gora — Behera "\u2195"
# Gora — Eskuma "\u21b3"
# Gora — Ezkerra "\u21b5"
# Behera — Eskuma "\u21b1"
# Behera — Ezkerra "\u21b0"
# Ezkerra — Eskuma "\u2194"
# Karratu beltza "\u25fe"
# Karratu zuria "\u25a2"

def polbek(labirintoa, hiztegia, eg):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    M=matrizhutsa(labirintoa)
    akzioak=["Gorantz", "Beherantz", "Eskumara", "Ezkerrerantz"]
    amaierakoegoera= (35,0)
    #OHARRA: Latex—ek ez ditu sinboloak irakurtzen.
    #Alboan sartu beharreko sinboloak adierazten dira.
    b="" #Gora—eskuma—ezker sinboloa
    c="" #Behera—eskuma—ezker sinboloa
    d="" #Gora — behera— eskuma sinboloa
    e= "" #Gora— behera — ezkerra sinbolo

    dim=len(eg)
    bektorea=np.zeros((dim,1))
```

```
    indize=-1

    for i in range(zabalera):
        for j in range(luze):
            bal=-1000
            if (((i,j),"Gorantz") in hiztegia)
            or (((i,j),"Beherantz") in hiztegia)
            or (((i,j),"Eskumara") in hiztegia)
            or (((i,j),"Ezkerrerantz") in hiztegia):
                indize=indize+1
            for k in range(4):
                if ((i,j), akzioak[k]) in hiztegia and (i,j)!=amaierakoegoera:
                    a=hiztegia[((i,j), akzioak[k])]
                    if a>bal:

                        bal=a
                        bektorea[indize][0]=a

                        if akzioak[k]=="Gorantz":
                            M[i][j]="\u2191"
                        elif akzioak[k]=="Beherantz":
                            M[i][j]="\u2193"
                        elif akzioak[k]=="Eskumara":
                            M[i][j]="\u2192"
                        elif akzioak[k]=="Ezkerrerantz":
                            M[i][j]="\u2190"
                    elif a==bal:
                        if (M[i][j]=="\u2191" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u2193" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u2195"
                        elif (M[i][j]=="\u2191" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u2192" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u21b3"
                        elif (M[i][j]=="\u2191" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u21b5"
                        elif (M[i][j]=="\u2193" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u2192" and akzioak[k]=="Beherantz"):
                            M[i][j]="\u21b1"
                        elif (M[i][j]=="\u2193" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Beherantz"):
                            M[i][j]="\u21b0"
```

```
                        elif (M[i][j]=="\u2192" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Eskuinera"):
                            M[i][j]= "\u2194"

                        if (M[i][j]=="\u2195" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b3" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u21b1" and akzioak[k]=="Gorantz"):
                            M[i][j]= d
                        elif (M[i][j]=="\u21b5" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b3" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2194" and akzioak[k]=="Gorantz"):
                            M[i][j]= b
                        elif (M[i][j]=="\u21b0" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b1" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2194" and akzioak[k]=="Beherantz"):
                            M[i][j]= c
                        elif (M[i][j]=="\u21b5" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u2195" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Gorantz"):
                            M[i][j]= e


                        if (M[i][j]==b and akzioak[k]=="Beherantz")
                        or (M[i][j]==c and akzioak[k]=="Gorantz")
                        or (M[i][j]==d and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]==e and akzioak[k]=="Eskumara"):
                            M[i][j]="\u2725"

                elif labirintoa[i][j]==1:
                    M[i][j]= "\u25a0"

                elif (((i,j), akzioak[k]) in hiztegia) and
                ((i,j)==amaierakoegoera):
                    M[i][j]="\u25a1"
                    erref=-1000
                    b=hiztegia[((i,j), akzioak[k])]
                    if b>erref:
                        erref=b
                        bektorea[indize][0]=b

    return M, bektorea

#Politikak printeatzeko
def printPolitika(politikak):
```

```
    for i in range(len(politikak)):
        print(politikak[i])



## Hasierako V(s) bektorea hasieratzeko.
def hasierakoegoerak(labirintoa,egoerak):
    dimentsioa=len(egoerak)
    M=np.random.randint(1,20,(dimentsioa,1))

    return M



## Balio-iterazio algoritmoa
def ValueIteration(labirintoa, gamma):

    egoerahiztegia=egoeraguztiak(labirintoa)
    hasierakobektorea=hasierakoegoerak(labirintoa,egoerahiztegia)
    egoerensaria=sariakVI(labirintoa)
    egoeralis=egoeralista(labirintoa)
    #Q-ren balioak (egoera akzio bakoitzerako), ondoren balio
    maximoak ematen duen akzioa lortzeko
    lab=informazioaosatu(labirintoa)

    #Egoera posiblea den labirintoan


    ##P matrizea
    MatGora=MatrizeValItGora(labirintoa,egoerahiztegia)
    MatBehera=MatrizeValItBehera(labirintoa,egoerahiztegia)
    MatEskuina=MatrizeValItEskuina(labirintoa,egoerahiztegia)
    MatEzkerra=MatrizeValItEzkerra(labirintoa,egoerahiztegia)


    #Episodio kopurua
    yardatza=[]
    xardatza=[]
    cont=0
    episodioa=1

    while episodioa!=-1:

        bektoreGora= egoerensaria + gamma*np.matmul(MatGora,
```

```
hasierakobektorea)
for i in range(len(egoeralis)):
    akzioa="Gorantz"
    if (egoeralis[i], akzioa) in lab:
        lab[(egoeralis[i], akzioa)]=bektoreGora[i]

bektoreBehera=egoerensaria + gamma*np.matmul(MatBehera,
hasierakobektorea)
for i in range(len(egoeralis)):
    akzioa="Beherantz"
    if (egoeralis[i], akzioa) in lab:
        lab[(egoeralis[i], akzioa)]=bektoreBehera[i]

bektoreEskuina= egoerensaria + gamma*np.matmul(MatEskuina,
hasierakobektorea)
for i in range(len(egoeralis)):
    akzioa="Eskumara"
    if (egoeralis[i], akzioa) in lab:
        lab[(egoeralis[i], akzioa)]=bektoreEskuina[i]

bektoreEzkerra= egoerensaria + gamma*np.matmul(MatEzkerra,
hasierakobektorea)
for i in range(len(egoeralis)):
    akzioa="Ezkerrerantz"
    if (egoeralis[i], akzioa) in lab:
        lab[(egoeralis[i], akzioa)]= bektoreEzkerra[i]


v0=hasierakobektorea
#Episodioaren egoera bakoitzaren balio-funtzioen kalkulua
+ egoeren politika
politikak, bek= polbek(labirintoa,lab,egoerahiztegia)
v1=bek


#Noiz gelditzeko zehazteko
puntua=float(max(v1-v0))
if puntua<0.001:
    episodioa=-1
    #Politikak printeatzeko
    printPolitika(politikak)

#Grafikoa egiteko x eta y aldatzen balioak
yardatza.append(puntua)
```

```
        xardatza.append(cont)

        #Kalkulatutako bektorea 'hasierakobektorean' bihurtzen da.
        hasierakobektorea=bek

        #Hurrengo episodioa
        cont=cont+1
        print(cont)

    #Grafikoa egiteko
    plt.plot(xardatza, yardatza)
    plt.xlabel('Iterazio kopurua')
    plt.ylabel('max(V_(n+1)-V_(n))')
    plt.title('Balio-iterazio algoritmoaren konbergentzia')
    plt.show()

    return lab
```

Hauek dira erabilitako funtzioak lortutako emaitzatik abiatuz, hasierako egoratik amaierako egoerara ailegatzeko ibilbide optimo bat lortzeko:

```
def akzioposiblea(egoera, labirintoa, hiztegia):
    errenkada= egoera[0]
    zutabea=egoera[1]
    akzioak=["Gorantz", "Beherantz", "Eskumara", "Ezkerrerantz"]
    akziopos=[]
    for i in range(4):
        if ((errenkada, zutabea), akzioak[i]) in hiztegia:
            akziopos.append(akzioak[i])

    return akziopos

def politikaoptimoa(egoera, labirintoa, hiztegia):
    akzioak=akzioposiblea(egoera,labirintoa,hiztegia)
    maximoa=-1000
    for i in range(len(akzioak)):
        if maximoa<=hiztegia[(egoera, akzioak[i])]:
            maximoa=hiztegia[(egoera, akzioak[i])]
            akziooptimoa=akzioak[i]

    return akziooptimoa

def ingurunea(labirintoa,egoera,ak):
    i=egoera[0] # Lehen elementua
    j=egoera[1] # Bigarren elementua
```

```
    if ak=="Gorantz":
        egoeraberria=(i-1,j)
        akzioa=1
    elif ak=="Eskumara":
        egoeraberria=(i,j+1)
        akzioa=2
    elif ak=="Beherantz":
        egoeraberria=(i+1,j)
        akzioa=3
    elif ak=="Ezkerrerantz":
        egoeraberria=(i,j-1)
        akzioa=4
    return egoeraberria,akzioa


def ibilbideoptimoa(labirintoa,gamma):
    hiztegioptimoa=ValueIteration(labirintoa,gamma)
    hasierakoegoera=(37,0)
    amaierakoegoera=(35,0)
    ibilbideoptimoa={}
    listaegoerak=[]
    while hasierakoegoera!=amaierakoegoera:
        akzoptimoa=politikaoptimoa(hasierakoegoera,labirintoa,hiztegioptimoa)
        listaegoerak.append(hasierakoegoera)
        Negoera,Nakzioa= ingurunea(labirintoa,hasierakoegoera,akzoptimoa)
        ibilbideoptimoa[hasierakoegoera]=Nakzioa
        hasierakoegoera=Negoera

    return ibilbideoptimoa, listaegoerak
```

# E. eranskina

# Q-learning algoritmoa

Hurrengo orrialdeetan *Pyhton*-en bitartez egindako Q-learning algoritmoa dago. Bai algoritmoan erabilitako funtzio osagarriak daude eta bai algoritmoa bera. Funtzio hauen bitartez, balio-iterazioa erabiliz, Villapresenteko labirintoaren hasierako egoeraren eta amaierako egoeraren arteko ibilbide optimoa lortu dugu.

---

```
import math
import numpy
import numpy as np
import time

from pylab import *
import matplotlib.pyplot as plt
import random


# Labirintoaren egoera/akzio bikote posible guztiak hiztegi batean bilduta.
# Q(s,a)= random hasieratuta
# Itxura: {((0,0), 'Beherantz'): float)}
#(egoera,akzioa) posiblea bada mug=0 eta ez bada posiblea mug=1.

def egoeraposiblea(egoera, labirintoa, ak):
    i=egoera[0] # Lehen elementua
    j=egoera[1] # Bigarren elementua
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    mug=0
    if ak=="Gorantz":
        if i-1<0 or labirintoa[i-1][j]==1:
            mug=1
```

```
    elif ak=="Beherantz":
        if i+1> (zabalera-1)  or labirintoa[i+1][j]==1:
            mug=1

    elif ak=="Eskumara":
        if  j+1> (luze-1) or labirintoa[i][j+1]==1:
            mug=1

    elif ak=="Ezkerrerantz":
        if j-1<0 or labirintoa[i][j-1]==1:
            mug=1

    return mug



def informazioaosatu(labirintoa):
    infor={}
    posfinala=(35,0)
    for i in range(len(labirintoa)):
        for j in range(len(labirintoa[0])):
            if labirintoa[i][j]!=1:
                for k in [1,2,3,4]:
                    if (i,j)!=posfinala:
                        if k==1:
                            ak="Gorantz"
                            if egoeraposiblea((i,j), labirintoa, ak)==0:
                                infor[(i,j), ak]=random.uniform(0,5)

                        if k==2:
                            ak="Beherantz"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=random.uniform(0,5)
                        if k==3:
                            ak="Eskumara"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=random.uniform(0,5)
                        if k==4:
                            ak="Ezkerrerantz"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=random.uniform(0,5)
                    else:
                        if k==1:
                            ak="Gorantz"
```

```
                            if egoeraposiblea((i,j), labirintoa, ak)==0:
                                infor[(i,j), ak]=0

                        if k==2:
                            ak="Beherantz"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=0
                        if k==3:
                            ak="Eskumara"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=0
                        if k==4:
                            ak="Ezkerrerantz"
                            if egoeraposiblea((i,j), labirintoa, ak)== 0:
                                infor[(i,j), ak]=0


    return infor



# Sarien matrizea.
# Labirintoaren matrizeen saria. Amaierako egoeraren saria +500
# izango da eta beste egoera guztien saria —1. Hormak = None.

def matrizesaria(labirintoa):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    sariak=numpy.zeros(shape=(zabalera,luze))
    posfinala=(35,0)
    for i in range(len(labirintoa)):
        for j in range(len(labirintoa[0])):
            if labirintoa[i][j]==0 and (i,j)!=posfinala:
                sariak[i][j]=—1
            elif labirintoa[i][j]==0 and (i,j)==posfinala:
                sariak[i][j]=500
            else:
                sariak[i][j]=None
    return sariak



# Akzio posibleak (legalak)
```

```
def akzioposiblea(egoera, labirintoa, hiztegia):
    errenkada= egoera[0]
    zutabea=egoera[1]
    akzioak=["Gorantz", "Beherantz", "Eskumara", "Ezkerrerantz"]
    akziopos=[]
    for i in range(4):
        if ((errenkada, zutabea), akzioak[i]) in hiztegia:
            akziopos.append(akzioak[i])

    return akziopos



# Epsilon— greedy estrategia

# Akzio 'optimoa' (Qbalioa handiena duena). Argmax Q(s,a)
def akzio_epsilongreedy(egoera, hiztegia, labirintoa):
    akzioposibleak=akzioposiblea(egoera,labirintoa,hiztegia)
    akziogreedy= []
    aux=—10000
    for i in range(len(akzioposibleak)):
        if (egoera, akzioposibleak[i]) in hiztegia:
            if hiztegia[(egoera, akzioposibleak[i])]>aux:
                aux=hiztegia[(egoera, akzioposibleak[i])]
                akziogreedy.append(akzioposibleak[i])
    akziooptimoa=akziogreedy[len(akziogreedy)—1]

    return akziooptimoa

# Akzio 'random' (Edozein akzio posible)
def randomakzioa (egoera,labirintoa,hiztegia):
    akzioposibleak=akzioposiblea(egoera,labirintoa,hiztegia)
    randomakzioa= random.choice(akzioposibleak) # epsilon/m

    return randomakzioa


# EPSILON—GREEDY
def epsilongreedy (egoera,hiztegia,epsilon, labirintoa):
    r=random.random()
    if r<epsilon:
        akzioa=randomakzioa(egoera,labirintoa,hiztegia)
```

```
    else:
        akzioa=akzio_epsilongreedy(egoera, hiztegia, labirintoa)

    return akzioa


# Ingurunea

#Ingurunea behatu eta egoera aldatu: egoera berria finkatu jakinda
#akzioa zein den (epsilon-greedy erabiliz) eta saria erakutsi

def ingurunea(egoera, akzioa, labirintoa,sariak):
    errenkada=egoera[0]
    zutabea=egoera[1]
    for i in range(4):
        if akzioa == "Gorantz":
            egoeraberria=(errenkada-1,zutabea)

        elif akzioa == "Beherantz":
            egoeraberria=(errenkada+1,zutabea)

        elif akzioa == "Eskumara":
            egoeraberria=(errenkada,zutabea+1)

        elif akzioa == "Ezkerrerantz":
            egoeraberria=(errenkada,zutabea-1)


    saria= sariak[egoeraberria[0]][egoeraberria[1]]

    return (egoeraberria, saria)

#Egoera posible guztien lista

def egoeralista(labirintoa):
    egoeralis=[]
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    kont=0
    for i in range(zabalera):
        for j in range(luze):
            if labirintoa[i][j]==0:
                egoeralis.append((i,j))
```

```
    return egoeralis

#########################################################
#Politika marrazteko

# Eskuma—Ezkerra—Gora—Behera "\u2725"
# Eskuma "\u2192"
# Ezkerra "\u2190"
# Gora "\u2191"
# Behera "\u2193"
# Gora — Behera "\u2195"
# Gora — Eskuma "\u21b3"
# Gora — Ezkerra "\u21b5"
# Behera — Eskuma "\u21b1"
# Behera — Ezkerra "\u21b0"
# Ezkerra — Eskuma "\u2194"
# Karratu beltza "\u25fe"
# Karratu zuria "\u25a2"

# Hurrengo funtzioan erabiltzeko
def matrizhutsa(labirintoa):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    M=[]
    for i in range(zabalera):
        m=[]
        for j in range(luze):
            m.append(0)
        M.append(m)
    return M

# Politiken mapa marrazteko
def matrizPolitika(labirintoa, hiztegia):
    zabalera= len(labirintoa)
    luze= len(labirintoa[0])
    M=matrizhutsa(labirintoa)
    akzioak=["Gorantz", "Beherantz", "Eskumara", "Ezkerrerantz"]
    amaierakoegoera=(35,0)
    #OHARRA: Latex—ek ez ditu sinboloak irakurtzen.
    #Alboan sartu beharreko sinboloak adierazten dira.
    b="" #Gora—eskuma—ezker sinboloa
    c="" #Behera—eskuma—ezker sinboloa
    d="" #Gora — behera— eskuma sinboloa
    e= "" #Gora— behera — ezkerra sinbolo
```

```
    for i in range(zabalera):
        for j in range(luze):
            bal=-1000
            for k in range(4):
                if ((i,j), akzioak[k]) in hiztegia and
                (i,j)!=amaierakoegoera:
                    a=hiztegia[((i,j), akzioak[k])]
                    if a>bal:
                        bal=a
                        if akzioak[k]=="Gorantz":
                            M[i][j]="\u2191"
                        elif akzioak[k]=="Beherantz":
                            M[i][j]="\u2193"
                        elif akzioak[k]=="Eskumara":
                            M[i][j]="\u2192"
                        elif akzioak[k]=="Ezkerrerantz":
                            M[i][j]="\u2190"
                    elif a==bal:
                        if (M[i][j]=="\u2191" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u2193" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u2195"
                        elif (M[i][j]=="\u2191" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u2192" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u21b3"
                        elif (M[i][j]=="\u2191" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Gorantz"):
                            M[i][j]="\u21b5"
                        elif (M[i][j]=="\u2193" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u2192" and akzioak[k]=="Beherantz"):
                            M[i][j]="\u21b1"
                        elif (M[i][j]=="\u2193" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Beherantz"):
                            M[i][j]="\u21b0"

                        elif (M[i][j]=="\u2192" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Eskuinera"):
                            M[i][j]= "\u2194"

                        if (M[i][j]=="\u2195" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b3" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u21b1" and akzioak[k]=="Gorantz"):
                            M[i][j]= d
                        elif (M[i][j]=="\u21b5" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b3" and akzioak[k]=="Ezkerrerantz")
```

```
                        or (M[i][j]=="\u2194" and akzioak[k]=="Gorantz"):
                            M[i][j]= b
                        elif (M[i][j]=="\u21b0" and akzioak[k]=="Eskumara")
                        or (M[i][j]=="\u21b1" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2194" and akzioak[k]=="Beherantz"):
                            M[i][j]= c
                        elif (M[i][j]=="\u21b5" and akzioak[k]=="Beherantz")
                        or (M[i][j]=="\u2195" and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]=="\u2190" and akzioak[k]=="Gorantz"):
                            M[i][j]= e


                        if (M[i][j]==b and akzioak[k]=="Beherantz")
                        or (M[i][j]==c and akzioak[k]=="Gorantz")
                        or (M[i][j]==d and akzioak[k]=="Ezkerrerantz")
                        or (M[i][j]==e and akzioak[k]=="Eskumara"):
                            M[i][j]="\u2725"

                elif labirintoa[i][j]==1:
                    M[i][j]= "\u25a0"
                elif (((i,j), akzioak[k]) in hiztegia) and
                ((i,j)==amaierakoegoera):
                    M[i][j]="\u25a1"


    return M


# Politiken matrizea printeatzeko
def printPolitika(politikak):
    for i in range(len(politikak)):
        print(politikak[i])

# Q_learning algoritmoan Q(estatua, akzioa) berritzeko kalkulatu
#beharreko balioa.
def maxbalAkzioak(egoera,hiztegia,labirintoa):
    akzioak= akzioposiblea(egoera, labirintoa,hiztegia)
    maxQ=-10000
    for i in range(len(akzioak)):
        a=hiztegia[(egoera, akzioak[i])]
        if a>maxQ:
            maxQ=a
    return maxQ
```

```
# Q-learning algoritmoa
def Q_learning(labirintoa, lab, egoerak, sariak, gamma, epsilon, ultepi,N):
    errenkadak= len(labirintoa)
    amaierakoegoera=(35,0)
    cont=0
    egoera=random.choice(egoerak)
    ft=False
    while egoera!=amaierakoegoera and cont!=800:
        akzioa=epsilongreedy(egoera, lab, epsilon, labirintoa)
        egoeraberria, saria =ingurunea(egoera,akzioa,labirintoa,sariak)
        if egoeraberria==amaierakoegoera:
            ft=True
        qbalioa=lab[(egoera,akzioa)]
        qbalioaber= qbalioa + 0.1*(saria +
        gamma*maxbalAkzioak(egoeraberria,lab,labirintoa)- qbalioa)
        lab={**lab,(egoera,akzioa): qbalioaber}
        egoera=egoeraberria
        cont=cont+1

    return lab,ft

# Ebaluazioa
# Ziklo bat (1000 episodio) egin ondoren ebaluaketa egiteko
def ebaluatu(labirintoa,gamma,lab,egoerak,sariak):
    errenkadak= len(labirintoa)
    amaierakoegoera=(35,0)
    cont=0
    egoera=random.choice(egoerak)
    falsetrue=False
    while egoera!=amaierakoegoera and cont!=800:
        akzioa=akzio_epsilongreedy(egoera, lab,labirintoa)
        egoeraberria=ingurunea(egoera,akzioa,labirintoa,sariak)[0]
        if egoeraberria==amaierakoegoera:
            falsetrue=True
        egoera=egoeraberria
        cont=cont+1

    return falsetrue

#Balio-iterazioaren bitartez lortutako emaitza kontuan hartuz,
#ibilbideoptimo funtzioak itzultzen duen
#ibilbide optimo bat - ibilbide optimo horren egoerak

def politikaoptimoa(egoera, labirintoa, hiztegia):
```

```
    akzioak=akzioposiblea(egoera,labirintoa,hiztegia)
    maximoa=-1000
    for i in range(len(akzioak)):
        if maximoa<hiztegia[(egoera, akzioak[i])]:
            maximoa=hiztegia[(egoera, akzioak[i])]
            akziooptimoa=akzioak[i]

    return akziooptimoa

def etiketa(ak):
    if ak=="Gorantz":
        akzioa=1
    elif ak=="Eskumara":
        akzioa=2
    elif ak=="Beherantz":
        akzioa=3
    elif ak=="Ezkerrerantz":
        akzioa=4

    return akzioa

def konprobatu(labirintoa,hiztegia):
    egoeraoptimoak=[(37, 0), (37, 1), (37, 2), (37, 3), (37, 4),
    (37, 5), (38, 5), (39, 5), (39, 6), (39, 7), (39, 8), (39, 9),
    (39, 10), (39, 11), (39, 12), (39, 13), (39, 14), (39, 15),
    (38, 15), (37, 15), (36, 15), (35, 15), (35, 16), (35, 17),
    (35, 18), (35, 19), (34, 19), (33, 19), (33, 18), (33, 17),
    (32, 17), (31, 17), (31, 18), (31, 19), (31, 20), (31, 21),
    (32, 21), (33, 21), (33, 22), (33, 23), (34, 23), (35, 23),
    (36, 23), (37, 23), (38, 23), (39, 23), (40, 23), (41, 23),
    (41, 22), (41, 21), (41, 20), (41, 19), (42, 19), (43, 19),
    (43, 20), (43, 21), (44, 21), (45, 21), (46, 21), (47, 21),
    (47, 20), (47, 19), (47, 18), (47, 17), (46, 17), (45, 17),
    (44, 17), (43, 17), (42, 17), (41, 17), (41, 16), (41, 15),
    (41, 14), (41, 13), (41, 12), (41, 11), (41, 10), (41, 9),
    (41, 8), (41, 7), (41, 6), (41, 5), (42, 5), (43, 5), (43, 6),
    (43, 7), (44, 7), (45, 7), (46, 7), (47, 7), (48, 7), (49, 7),
    (50, 7), (51, 7), (52, 7), (53, 7), (53, 8), (53, 9), (52, 9),
    (51, 9), (51, 10), (51, 11), (51, 12), (51, 13), (51, 14),
    (51, 15), (51, 16), (51, 17), (52, 17), (53, 17), (54, 17),
    (55, 17), (56, 17), (57, 17), (57, 18), (57, 19), (57, 20),
    (57, 21), (57, 22), (57, 23), (58, 23), (59, 23), (59, 24),
    (59, 25), (59, 26), (59, 27), (59, 28), (59, 29), (59, 30),
    (59, 31), (59, 32), (59, 33), (58, 33), (57, 33), (56, 33),
```

```
(55, 33), (54, 33), (53, 33), (53, 32), (53, 31), (53, 30),
(53, 29), (53, 28), (53, 27), (52, 27), (51, 27), (50, 27),
(49, 27), (49, 28), (49, 29), (50, 29), (51, 29), (51, 30),
(51, 31), (51, 32), (51, 33), (51, 34), (51, 35), (51, 36),
(51, 37), (51, 38), (51, 39), (52, 39), (53, 39), (53, 40),
(53, 41), (53, 42), (53, 43), (53, 44), (54, 44), (55, 44),
(55, 45), (56, 45), (57, 45), (58, 45), (59, 45), (59, 44),
(59, 43), (59, 42), (59, 41), (60, 41), (61, 41), (61, 42),
(61, 43), (61, 44), (61, 45), (62, 45), (63, 45), (63, 46),
(63, 47), (63, 48), (63, 49), (64, 49), (65, 49), (65, 48),
(65, 47), (65, 46), (65, 45), (65, 44), (65, 43), (65, 42),
(65, 41), (66, 41), (67, 41), (68, 41), (69, 41), (69, 42),
(69, 43), (69, 44), (69, 45), (68, 45), (67, 45), (67, 46),
(67, 47), (68, 47), (69, 47), (69, 48), (69, 49), (69, 50),
(69, 51), (69, 52), (69, 53), (69, 54), (69, 55), (68, 55),
(67, 55), (67, 54), (67, 53), (67, 52), (67, 51), (66, 51),
(65, 51), (65, 52), (65, 53), (65, 54), (65, 55), (65, 56),
(65, 57), (66, 57), (67, 57), (67, 58), (67, 59), (67, 60),
(67, 61), (66, 61), (65, 61), (64, 61), (63, 61), (62, 61),
(61, 61), (61, 60), (61, 59), (61, 58), (61, 57), (61, 56),
(61, 55), (61, 54), (61, 53), (61, 52), (61, 51), (61, 50),
(61, 49), (61, 48), (61, 47), (60, 47), (59, 47), (58, 47),
(57, 47), (56, 47), (55, 47), (54, 47), (53, 47), (52, 47),
(51, 47), (50, 47), (49, 47), (48, 47), (47, 47), (47, 46),
(47, 45), (48, 45), (49, 45), (49, 44), (49, 43), (48, 43),
(47, 43), (46, 43), (45, 43), (44, 43), (43, 43), (42, 43),
(41, 43), (40, 43), (39, 43), (38, 43), (37, 43), (36, 43),
(35, 43), (34, 43), (33, 43), (33, 42), (33, 41), (34, 41),
(35, 41), (35, 40), (35, 39), (34, 39), (33, 39), (32, 39),
(31, 39), (31, 40), (31, 41), (31, 42), (31, 43), (30, 43),
(29, 43), (28, 43), (27, 43), (26, 43), (25, 43), (25, 42),
(25, 41), (25, 40), (25, 39), (25, 38), (25, 37), (25, 36),
(25, 35), (24, 35), (23, 35), (22, 35), (21, 35), (21, 36),
(21, 37), (21, 38), (21, 39), (21, 40), (21, 41), (21, 42),
(21, 43), (20, 43), (19, 43), (18, 43), (17, 43), (16, 43),
(15, 43), (15, 42), (15, 41), (15, 40), (15, 39), (16, 39),
(17, 39), (17, 38), (17, 37), (16, 37), (15, 37), (15, 36),
(15, 35), (15, 34), (15, 33), (15, 32), (15, 31), (15, 30),
(15, 29), (15, 28), (15, 27), (16, 27), (17, 27), (17, 26),
(17, 25), (16, 25), (15, 25), (15, 24), (15, 23), (15, 22),
(15, 21), (15, 20), (15, 19), (16, 19), (17, 19), (18, 19),
(19, 19), (20, 19), (21, 19), (21, 18), (21, 17), (21, 16),
(21, 15), (22, 15), (23, 15), (24, 15), (25, 15), (26, 15),
(27, 15), (28, 15), (29, 15), (30, 15), (31, 15), (32, 15),
```

```
(33, 15), (33, 14), (33, 13), (33, 12), (33, 11), (33, 10),
(33, 9), (33, 8), (33, 7), (33, 6), (33, 5), (34, 5), (35, 5),
(35, 4), (35, 3), (35, 2), (35, 1)]
ibilbideoptimoa={(37, 0): 2, (37, 1): 2, (37, 2): 2,
(37, 3): 2, (37, 4): 2, (37, 5): 3, (38, 5): 3, (39, 5): 2,
(39, 6): 2, (39, 7): 2, (39, 8): 2, (39, 9): 2, (39, 10): 2,
(39, 11): 2, (39, 12): 2, (39, 13): 2, (39, 14): 2, (39, 15): 1,
(38, 15): 1, (37, 15): 1, (36, 15): 1, (35, 15): 2, (35, 16): 2,
(35, 17): 2, (35, 18): 2, (35, 19): 1, (34, 19): 1, (33, 19): 4,
(33, 18): 4, (33, 17): 1, (32, 17): 1, (31, 17): 2, (31, 18): 2,
(31, 19): 2, (31, 20): 2, (31, 21): 3, (32, 21): 3, (33, 21): 2,
(33, 22): 2, (33, 23): 3, (34, 23): 3, (35, 23): 3, (36, 23): 3,
(37, 23): 3, (38, 23): 3, (39, 23): 3, (40, 23): 3, (41, 23): 4,
(41, 22): 4, (41, 21): 4, (41, 20): 4, (41, 19): 3, (42, 19): 3,
(43, 19): 2, (43, 20): 2, (43, 21): 3, (44, 21): 3, (45, 21): 3,
(46, 21): 3, (47, 21): 4, (47, 20): 4, (47, 19): 4, (47, 18): 4,
(47, 17): 1, (46, 17): 1, (45, 17): 1, (44, 17): 1, (43, 17): 1,
(42, 17): 1, (41, 17): 4, (41, 16): 4, (41, 15): 4, (41, 14): 4,
(41, 13): 4, (41, 12): 4, (41, 11): 4, (41, 10): 4, (41, 9): 4,
(41, 8): 4, (41, 7): 4, (41, 6): 4, (41, 5): 3, (42, 5): 3,
(43, 5): 2, (43, 6): 2, (43, 7): 3, (44, 7): 3, (45, 7): 3,
(46, 7): 3, (47, 7): 3, (48, 7): 3, (49, 7): 3, (50, 7): 3,
(51, 7): 3, (52, 7): 3, (53, 7): 2, (53, 8): 2, (53, 9): 1,
(52, 9): 1, (51, 9): 2, (51, 10): 2, (51, 11): 2, (51, 12): 2,
(51, 13): 2, (51, 14): 2, (51, 15): 2, (51, 16): 2, (51, 17): 3,
(52, 17): 3, (53, 17): 3, (54, 17): 3, (55, 17): 3, (56, 17): 3,
(57, 17): 2, (57, 18): 2, (57, 19): 2, (57, 20): 2, (57, 21): 2,
(57, 22): 2, (57, 23): 3, (58, 23): 3, (59, 23): 2, (59, 24): 2,
(59, 25): 2, (59, 26): 2, (59, 27): 2, (59, 28): 2, (59, 29): 2,
(59, 30): 2, (59, 31): 2, (59, 32): 2, (59, 33): 1, (58, 33): 1,
(57, 33): 1, (56, 33): 1, (55, 33): 1, (54, 33): 1, (53, 33): 4,
(53, 32): 4, (53, 31): 4, (53, 30): 4, (53, 29): 4, (53, 28): 4,
(53, 27): 1, (52, 27): 1, (51, 27): 1, (50, 27): 1, (49, 27): 2,
(49, 28): 2, (49, 29): 3, (50, 29): 3, (51, 29): 2, (51, 30): 2,
(51, 31): 2, (51, 32): 2, (51, 33): 2, (51, 34): 2, (51, 35): 2,
(51, 36): 2, (51, 37): 2, (51, 38): 2, (51, 39): 3, (52, 39): 3,
(53, 39): 2, (53, 40): 2, (53, 41): 2, (53, 42): 2, (53, 43): 2,
(53, 44): 3, (54, 44): 3, (55, 44): 2, (55, 45): 3, (56, 45): 3,
(57, 45): 3, (58, 45): 3, (59, 45): 4, (59, 44): 4, (59, 43): 4,
(59, 42): 4, (59, 41): 3, (60, 41): 3, (61, 41): 2, (61, 42): 2,
(61, 43): 2, (61, 44): 2, (61, 45): 3, (62, 45): 3, (63, 45): 2,
(63, 46): 2, (63, 47): 2, (63, 48): 2, (63, 49): 3, (64, 49): 3,
(65, 49): 4, (65, 48): 4, (65, 47): 4, (65, 46): 4, (65, 45): 4,
(65, 44): 4, (65, 43): 4, (65, 42): 4, (65, 41): 3, (66, 41): 3,
```

```
(67, 41): 3, (68, 41): 3, (69, 41): 2, (69, 42): 2, (69, 43): 2,
(69, 44): 2, (69, 45): 1, (68, 45): 1, (67, 45): 2, (67, 46): 2,
(67, 47): 3, (68, 47): 3, (69, 47): 2, (69, 48): 2, (69, 49): 2,
(69, 50): 2, (69, 51): 2, (69, 52): 2, (69, 53): 2, (69, 54): 2,
(69, 55): 1, (68, 55): 1, (67, 55): 4, (67, 54): 4, (67, 53): 4,
(67, 52): 4, (67, 51): 1, (66, 51): 1, (65, 51): 2, (65, 52): 2,
(65, 53): 2, (65, 54): 2, (65, 55): 2, (65, 56): 2, (65, 57): 3,
(66, 57): 3, (67, 57): 2, (67, 58): 2, (67, 59): 2, (67, 60): 2,
(67, 61): 1, (66, 61): 1, (65, 61): 1, (64, 61): 1, (63, 61): 1,
(62, 61): 1, (61, 61): 4, (61, 60): 4, (61, 59): 4, (61, 58): 4,
(61, 57): 4, (61, 56): 4, (61, 55): 4, (61, 54): 4, (61, 53): 4,
(61, 52): 4, (61, 51): 4, (61, 50): 4, (61, 49): 4, (61, 48): 4,
(61, 47): 1, (60, 47): 1, (59, 47): 1, (58, 47): 1, (57, 47): 1,
(56, 47): 1, (55, 47): 1, (54, 47): 1, (53, 47): 1, (52, 47): 1,
(51, 47): 1, (50, 47): 1, (49, 47): 1, (48, 47): 1, (47, 47): 4,
(47, 46): 4, (47, 45): 3, (48, 45): 3, (49, 45): 4, (49, 44): 4,
(49, 43): 1, (48, 43): 1, (47, 43): 1, (46, 43): 1, (45, 43): 1,
(44, 43): 1, (43, 43): 1, (42, 43): 1, (41, 43): 1, (40, 43): 1,
(39, 43): 1, (38, 43): 1, (37, 43): 1, (36, 43): 1, (35, 43): 1,
(34, 43): 1, (33, 43): 4, (33, 42): 4, (33, 41): 3, (34, 41): 3,
(35, 41): 4, (35, 40): 4, (35, 39): 1, (34, 39): 1, (33, 39): 1,
(32, 39): 1, (31, 39): 2, (31, 40): 2, (31, 41): 2, (31, 42): 2,
(31, 43): 1, (30, 43): 1, (29, 43): 1, (28, 43): 1, (27, 43): 1,
(26, 43): 1, (25, 43): 4, (25, 42): 4, (25, 41): 4, (25, 40): 4,
(25, 39): 4, (25, 38): 4, (25, 37): 4, (25, 36): 4, (25, 35): 1,
(24, 35): 1, (23, 35): 1, (22, 35): 1, (21, 35): 2, (21, 36): 2,
(21, 37): 2, (21, 38): 2, (21, 39): 2, (21, 40): 2, (21, 41): 2,
(21, 42): 2, (21, 43): 1, (20, 43): 1, (19, 43): 1, (18, 43): 1,
(17, 43): 1, (16, 43): 1, (15, 43): 4, (15, 42): 4, (15, 41): 4,
(15, 40): 4, (15, 39): 3, (16, 39): 3, (17, 39): 4, (17, 38): 4,
(17, 37): 1, (16, 37): 1, (15, 37): 4, (15, 36): 4, (15, 35): 4,
(15, 34): 4, (15, 33): 4, (15, 32): 4, (15, 31): 4, (15, 30): 4,
(15, 29): 4, (15, 28): 4, (15, 27): 3, (16, 27): 3, (17, 27): 4,
(17, 26): 4, (17, 25): 1, (16, 25): 1, (15, 25): 4, (15, 24): 4,
(15, 23): 4, (15, 22): 4, (15, 21): 4, (15, 20): 4, (15, 19): 3,
(16, 19): 3, (17, 19): 3, (18, 19): 3, (19, 19): 3, (20, 19): 3,
(21, 19): 4, (21, 18): 4, (21, 17): 4, (21, 16): 4, (21, 15): 3,
(22, 15): 3, (23, 15): 3, (24, 15): 3, (25, 15): 3, (26, 15): 3,
(27, 15): 3, (28, 15): 3, (29, 15): 3, (30, 15): 3, (31, 15): 3,
(32, 15): 3, (33, 15): 4, (33, 14): 4, (33, 13): 4, (33, 12): 4,
(33, 11): 4, (33, 10): 4, (33, 9): 4, (33, 8): 4, (33, 7): 4,
(33, 6): 4, (33, 5): 3, (34, 5): 3, (35, 5): 4, (35, 4): 4,
(35, 3): 4, (35, 2): 4, (35, 1): 4}
hiztegiberria=ibilbideoptimoa.copy()
```

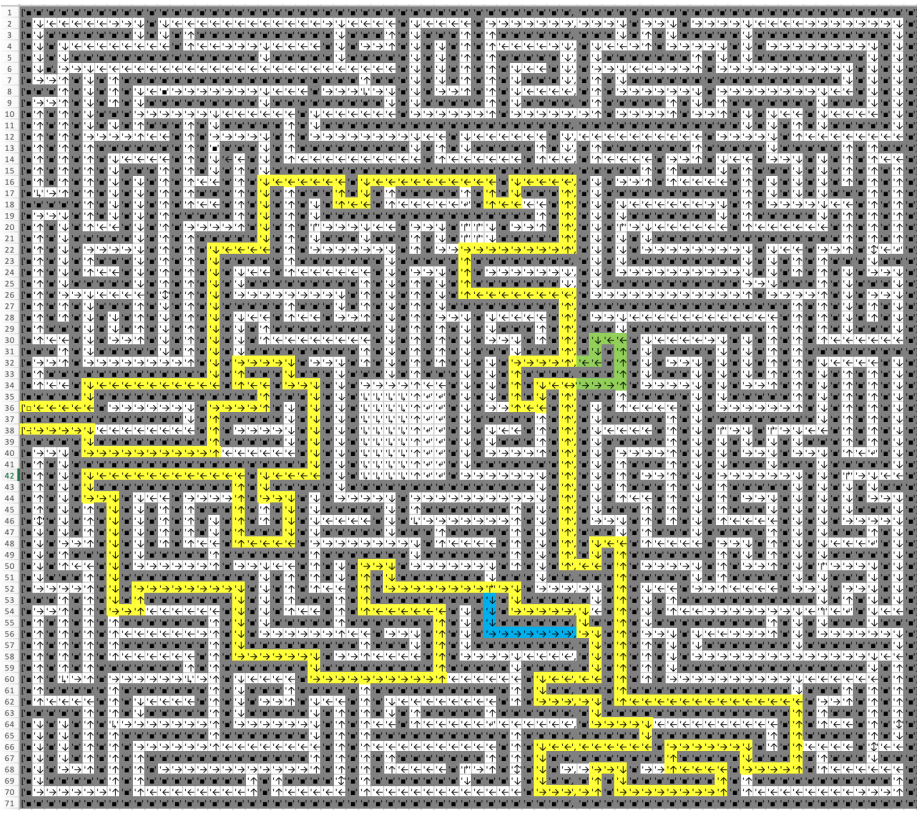
```
    for i in range(len(egoeraoptimoak)):
        akziooptimoa=politikaoptimoa(egoeraoptimoak[i],labirintoa,hiztegia)
        Nakziooptimoa=etiketa(akziooptimoa)
        hiztegiberria[egoeraoptimoak[i]]=Nakziooptimoa

    cont=0
    for i in range(len(egoeraoptimoak)):
        if (ibilbideoptimoa[egoeraoptimoak[i]]-hiztegiberria[egoeraoptimoak[i]])!=0:
            cont=cont+1


    return cont

def zeharkatu(labirintoa, gamma, epsilon):
    #Behin bakarrik eratzeko informazioa.
    sariak=matrizesaria(labirintoa)
    lab=informazioaosatu(labirintoa)
    egoerak=egoeralista(labirintoa)

    #Zikloentzako:
    N=1000 #Epsilon-greedy - Aldaketak egin
    M=100 #Akzio onena - Ez egin aldaketak

    ratioN=0
    ratioM=0
    Mkopurua=0
    zikloa=0
    yardatza=[]
    xardatza=[]
    zardatza=[]
    pos=-1
    a=0
    Mfinko=0

    while (pos,Mfinko)!=(0,30):
        print("Ziklo honetatik zoaz:",zikloa)
        eps=epsilon
        for n in range(N):
            labhiztegia, amaieraaurkituN =
            Q_learning(labirintoa,lab,egoerak,sariak, gamma, epsilon, n,N)
            lab=labhiztegia
            if amaieraaurkituN==True:
                ratioN=ratioN+1
```

```
        #Epsilon txikitu
        eps=eps*0.999
        eps=max(eps,0.1)

        if n%100==0:
            print("N hau da:",n)
            print(time.process_time())

        if n==999:
            print("N=1000-an:")
            print(time.process_time())

    ebaluazioN=ratioN/N
    print("N-ren ratioaren ebaluazioa", ebaluazioN)
    print("Kopuru honetan labirintotik irten da (N)",ratioN)

    for m in range(M):
        amaieraaurkituM=ebaluatu(labirintoa,gamma,lab,egoerak,sariak)
        if amaieraaurkituM==True:
            ratioM=ratioM+1

    ebaluazioM=ratioM/M
    if ebaluazioM==1:
        Mkopurua=Mkopurua+1

    if Mkopurua>=30:
        Mfinko=30 #Gerta daitekeelako Mkopurua=30 izanik egoera
                  #guztien politika optimoa ez aurkitzea

    print("M-ren ratioaren ebaluazioa:",ebaluazioM)
    print("Kopuru honetan labirintotik irten da (M)",ratioM)
    yardatza.append(ebaluazioM)

    pos=konprobatu(labirintoa,lab)
    zardatza.append(pos)
    if pos!=0 and a==0:
        print("Oraindik ez du egoera kopuru hauen politika
        optimoa aurkitu:",pos)

    if pos==0:
        a=1
        print("Ibilbide optimoaren poltika guztiak aurkitu ditu")

    ratioN=0
```

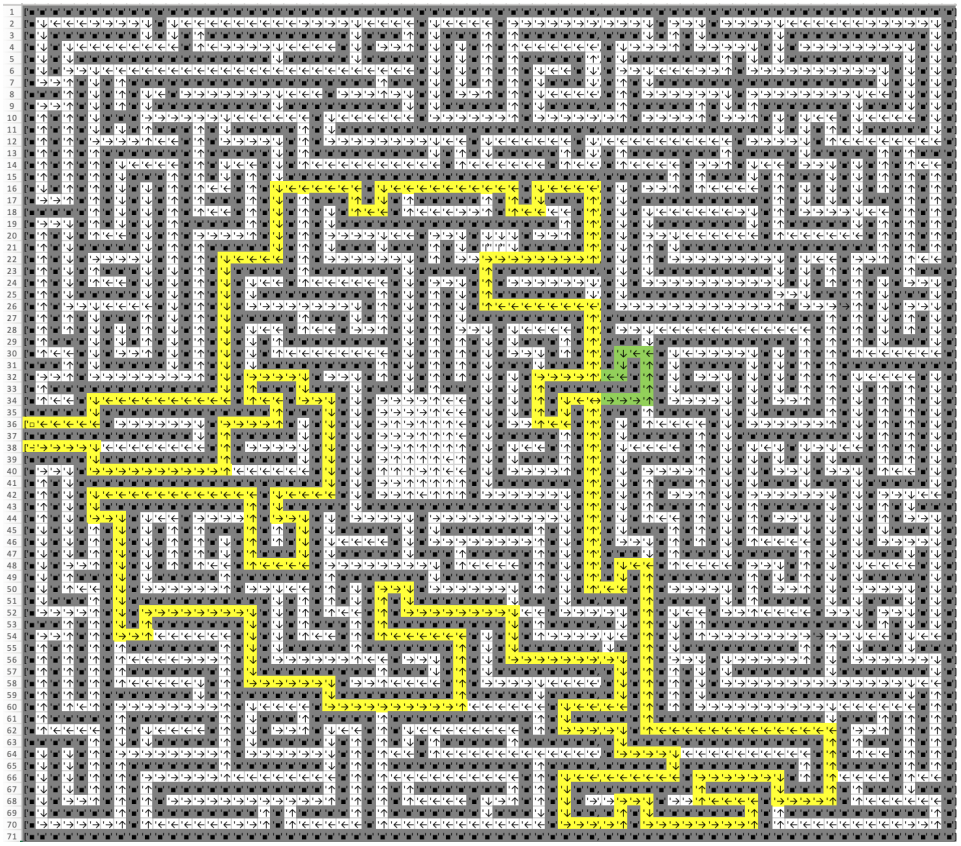
```
        ratioM=0
        zikloa=zikloa+1
        xardatza.append(zikloa)

    if Mfinko==30 and pos==0:
        politikak=matrizPolitika(labirintoa,lab)
        printPolitika(politikak)

    plt.plot(xardatza,yardatza,'o')
    plt.xlabel('Ziklo kopurua')
    plt.ylabel('Ebaluazioaren ratio')
    plt.title('Q-learning-en ebaluazioa')
    plt.show()

    plt.plot(xardatza,zardatza,'o')
    plt.xlabel('Ziklo kopurua'
    plt.ylabel('Politika optimoa ez duten egoeren kopurua')
    plt.title('Q-learning-en ebaluazioa balio-iterazioaren
    ibilbide optimoa jakinda')
    plt.show()

    return lab
```

# F. eranskina

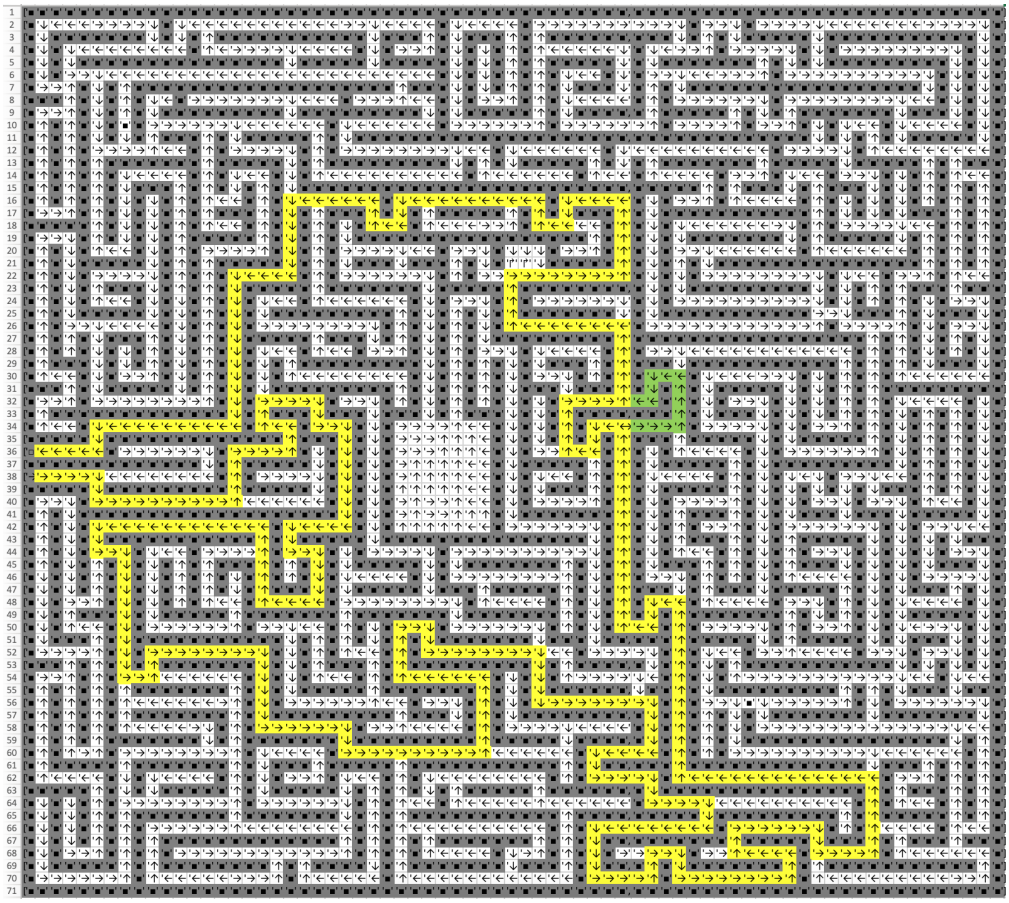
# Emaitzak



**F.1. irudia**: Balio-funtzioaren bidez lortutako politika optimoen mapa



**F.2. irudia**: Q-learning-en bidez lortutako politiken mapa (ratioa=1, 20 aldiz)



**F.3. irudia**: Q-learning-en bidez lortutako politiken mapa (ratioa=1, 30 aldiz + balio-iterazioa)

# Bibliografia


[1] Bagnato J.I., *Aprende Machine Learning en Español: Teoría + Práctica Python*, ISBN agentzia Espainian, 2020.

[2] Barto, A.G eta Sutton, R.S., *Reinforcement learning: An introduction*, MIT press, 1998.

[3] García Polo F.J., "Aprendizaje por Refuerzo para la Toma de Decisiones Segura en Dominios con Espacios de Estados y Acciones Continuos", Tesis doktorala, Madrilgo Karlos III.a Unibertsitatea, Madril, Leganés, 2012, 13.orr.

[4] Guerra C., "Inteligencia Artificial: Aprendizaje por refuerzo", Las Palmas Kanaria Handikoa Unibertsitatea, (`http://cayetanoguerra.github.io/ia/rl/Aprendizaje_por_refuerzo_apuntes.pdf`).

[5] Morales E.F.,Sucar L.E, *Procesos de Decisión de Markov y Aprendizaje por Refuerzo"*, 13.kapitulua, Sidorov (editorea), Alfaomega Grupo Editor, 2018.

[6] Rao A. , "Optimal policy from optimal value function", Standford CME 241, (`https://web.stanford.edu/class/cme241/lecture_slides/OptimalPolicyExistence.pdf`).

[7] Silva M., "Aprendizaje por Refuerzo: Procesos de Decisión de Markov - Parte 1", 2019 (`https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-procesos-de-decisi%C3%B3n-de-markov-parte-2-d219358ecd76`).

[8] Silver, D., *Lectures on Reinforcement Learning*, 2015 (`http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html`).

[9] Torres J., *Introducción al aprendizaje por refuerzo profundo: Teoría y práctica en Python*, Watch this space; lehenengo edizioa, 2021.

[10] Zico Kolter J., "15-780:Markov Decision Processes", 2016, (`http://www.cs.cmu.edu/afs/cs/academic/class/15780-s16/www/slides/mdps.pdf`).