eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

INFORMATIKA INGENIARITZAKO GRADUA
SOFTWARE INGENIARITZA

GRADU AMAIERAKO LANA

Diabetes gaixotasuna kudeatzeko kode irekiko aplikazio bati funtzionalitate berri bat gehitu: hartu beharreko insulina kopurua modu automatikoan kalkulatu errefortzu bidezko ikasketa erabiliz

Egilea

Unai Hurtado Pelaez

Zuzendariak

Iñigo Mendialdua, Iñigo Perona

2022-2023 Ikasturtea

Laburpena

Digitalizazioak garapen eta aldaketa handiak ekarri ditu gizarte mailan baita zientzian ere. Medikuntzan adibidez datuen kantitatea handitu eta tratamendu sistema berriak sortu dira hainbat gaixotasunen maneiua erraztuz, adibidez diabetesaren maneiua.

Diabetesa gaixotasun konplexua da eta bere maneiu desegokiak gaixoaren osasuna kaltetu dezake. Konkretuki gluzemia (edo odoleko azukrea) tarte osasuntsu batean mantendu behar da. "xDrip" diabetes maneiuan laguntzeko helburua duen kode irekiko Android proiektua da. xDrip-ek hainbat funtzio ditu, interesgarrienetariko bat *intsulina* gomendio sistema izanik.

Gomendio sistemak diabetikoek egunero egin behar duten kalkuluak errazten ditu. Kalkulu hauek gluzemia tarte osasuntsuan mantentzeko erabiltzen dira. Hala ere sistemak hainbat muga ditu. Proiektu honen helburua xDrip aplikaziori funtzionalitate berri bat gehitzea da, *Reinforcement Learning* (*RL*) erabiltzen duen intsulina gomendio sistema berri bat. *RL*-ak ingurune batean probak eginez ikasten duen adimen artifizialeko teknika bat da. Hainbat ikerketek *RL* erabiliz gaur egun erabiltzen diren diabetes intsulina gomendio sistemak hobetzea lortu dute. Beraz, sistema berri honekin xDrip-ek duen gomendio sistemaren mugak gainditzea lortu nahi da.

Aurkibidea

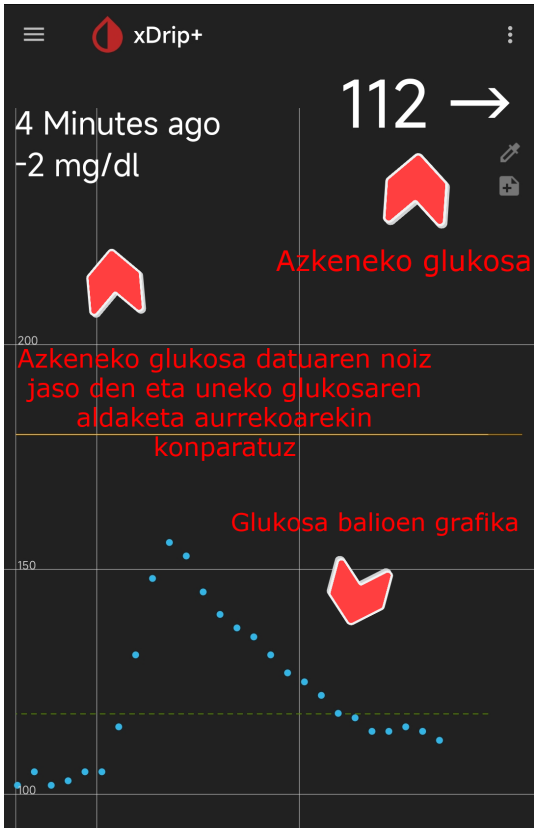
1 Sarrera

Informatikak, digitalizazioaren bidez, garapen eta aldaketa handiak ekarri ditu bai gizarte mailan eta baita zientzian ere, hala nola medikuntzan. Digitalizazioak datu kantitate handien meatzaritza eta prozesuen automatizazioa ekarri du. Gainera, konputazio ahalmenak ere gora egin du, eta honek, aplikazio konplexuagoak sortzeari bide eman die. Ondorioz, datu multzo eta konputazio ahalmen hauek gabe posibleak izango ez liratekeen sistemak sortu dira, hala nola AlphaFold [1] edo diabetiko baten *intsulina* injekzio lagundua.

Diabetesak (ikus 4.1. atala) garapen mediku eta informatiko hauetatik etekin handia lortu du, izan ere diabetesa maneiatzea oso zaila bihurtu daiteke. Diabetiko batek *intsulina* injektatu behar du egunero. Injekzio kantitatea formula bidez kalkulatzeko eta sendo funtzionatzeko, pertsona bakoitzaren gorputzaren araberako parametro asko ezagutu behar dira. Gainera *intsulina* beharrak ondo ez kontrolatzeak emaitza larriak ekar ditzake (organoen hutsegiteak, konorte-galera...). Hau guztia maneiatzen laguntzeko erremintak daude, "*xDrip*" Android aplikazioa adibidez.

xDrip (ikus 1.1. irudia) diabetesa maneiatzeko helburua duen kode irekiko Android proiektua da. Funtzionalitate asko ditu, hauen artean: gailu medikuen balioak (*gluzemia* edo odoleko azukrea adibidez) lortu eta era erosoan erakutsi (ikus 1.1. irudia), diabetesaren maneiuaren historiala gorde, jateko orduan egin behar diren kalkulu batzuk erraztu edo automatizatu...

Gradu Amaierako Lanerako *xDrip* aplikazioari hedapen bat egitea erabaki da. *xDrip*-en formula matematikoak aldagai gutxi hartzen ditu kontuan (erraz neurtu daitezkeenak) eta beraz emaitzak ez dira sendoak. Hedapenaren bidez Adimen Artifizialeko *Reinforcement Learning (RL)* teknikak erabiliko dira *intsulina* injekzio kalkulua egiteko. *RL* bidez *gluzemia* (odoleko azukrea) patroiak kontuan hartuko dira. Patroiek neurtu ezin diren aldagaien informazioa ematen dute eta beraz emaitza hobeak lortu daitezke. Hainbat ikerketek [2, 3] teknika hauen eraginkortasuna probatu dute.



Irudia 1.1: *xDrip*-en interfaze nagusia.

1.1 Proiektuaren deskribapena

Intsulina mendekotasuna duen diabetesaren maneiua ez da erreza. Helburua sinplea da, diabetiko baten *gluzemia* (azukre maila) tarte batean mantentzea da, gutxi gora-behera 80-180 mg/dl. *Gluzemia intsulina* bidez murriztu eta karbohidratoen bidez igotzen da. Janari gehienek karbohidratoak dituztenez diabetiko batek *intsulina* injekzioa beharko du gluzemia orekatzeko.

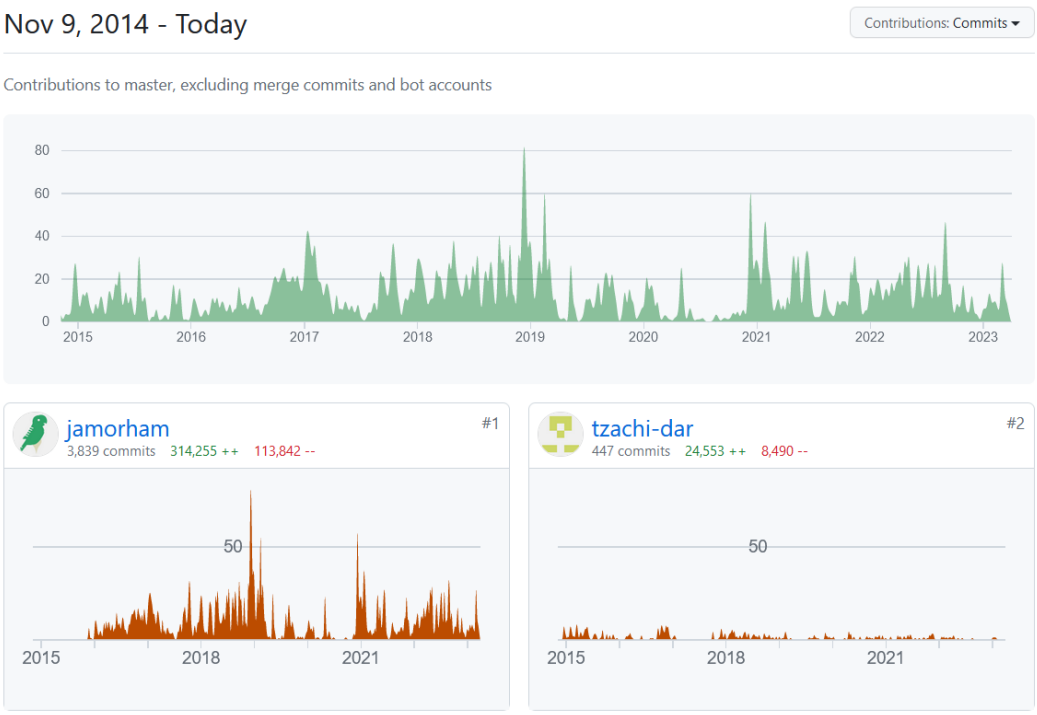
Jatean eman beharko den *intsulina* kalkulatzeko eta kalkulu hau guztiz zuzena izateko parametro asko kontuan hartu behar dira (karbohidratoak, ratioak, *intsulina* sentsibilitatea, *intsulina* gorputzean...). Zoritxarrez hainbat parametro ezin dira erraz lortu, adibidez estres maila. Ondorioz, kalkuluan parametro batzuk alde batera uzten dira zehaztasuna galduz.

Parametro erabilgarriak biltzeko eta kalkulua egiteko *xDrip* erabili daiteke baina kalkulua egiteko sistema ez da oso zehatza parametro gutxi batzuk bakarrik

erabiltzen dituelako. Sistema hau hobetzeko kalkulua malguagoa eta pertsonalizatuagoa izan behar da. Hau lortzeko asmoarekin, Adimen Artifizialaren *Reinforcement Learning* (*RL*) tekniken bidez sortutako gomendio sistemak erabiliko dituen funtzionalitate berri bat garatuko da.

1.2 xDrip-en egoera OSS bezala

xDrip aplikazioa Emma Black-ek sortu zuen 2014-an (DexDrip izenarekin). Ondoren "jamorham" ezizeneko garatzaileak xDrip+ sortu zuen. Gaur egun *xDrip*-en fork[1] erabiliena eta oinarrizkoa dena Nightscout[2] fundazioak maneiatzen du. Gaur egunera arte 7.573 *commit*, 113 *issue* (ireki) eta 59 *pull request* jaso ditu. Mundu osoko 73 garatzailek lan egin dute aplikazioan.



Irudia 1.2: *xDrip*-ek bere bizitza osoan izandako kontribuzioak. Behean bi garatzaile aktiboenak.

Proiektua Android aplikazio bat bezala hasi zen eta ondoren WearOS-entzako[3]

[1] https://github.com/NightscoutFoundation/xDrip
[2] http://www.nightscout.info/
[3] Google-en smartwatch-entzako sistema eragilea

aplikazio propioa gehitu zitzaion. Bi aplikazio izateak eta *xDrip*-ek duen funtzionalitate kantitate handiaren ondorioz, proiektuak Java kode lerro asko ditu, 150.000 baino gehiago.

Aplikazioaren kodearen egoera ez da oso ona. Softwarearen diseinu aldetik, egitura nahasgarria du, izan ere, kodea leku askotan errepikaturik baitago. Honek bere mantenigarritasuna zailtzen du eta dokumentaziorik ez duenez aplikazioaren garapena konplexua izan daiteke.

1.3 Proiektuaren motibazioak

Proiektua garatzeko motibazioak hurrengoak dira:

- **Proiektu baten aldaketa**: unibertsitatean egindako lan gehienak zerbait zerotik sortzean oinarritzen dira, kontzeptuak ikasteko era eraginkorra baita. Hala ere, enpresa batean lan egitean aplikazio asko jada sortuta daude eta kasu askotan garatzailearen lana aplikazioaren hedapena eta mantentzea izango da. Beraz GrAL-aren bidez aplikazioen hedapenean esperientzia lortu nahi da.

- **Proiektu handi bat**: unibertsitateko proiektuak tamainan txikiak izaten dira, sortu daitezkeen arazoak mugatuak izateko. Hauek gainera funtzionalitate murriztuak dituzte. Proiektu handietan aldaketak egitean arazo asko sortu daitezke, hala nola bateraezintasuna. Bestetik, *xDrip*-ek ez du ia dokumentaziorik eta ondorioz, aplikazioa ulertzea erronka bat izango da.

- **Erabilgarritasuna**: garatuko den funtzionalitatea *xDrip*-en integratzeko aukera dago, eta beraz beste edozein pertsonak erabili ahalko du. Gainera interes pertsonala dut, izan ere diabetikoa naizenez aplikazioa egunero erabiltzen dut.

- **Teknologia berriak erabiltzeko aukera**: *xDrip*-ren hedapena garatzeko orduan ikasketetan erabili ez diren teknologiak jorratuko dira, adibidez Android garapenean beharrezkoak diren erremintak. Gainera, nire kasuan, "Advanced Artificial Intelligence Techniques" irakasgaia egin eta *RL* ezagutu nuen. Teknologia honetan interesa sortu zitzaidan eta gainera mundu errealeko hainbat atazetan *RL*-ak emaitza oso onak lortu ditu, beraz hau diabetesa maneiatzeko laguntza handikoa izan ahal zela pentsatu nuen.

2 Planifikazioa

2.1 Irismena

GrAL guztien helburua unibertsitatean lortutako ezagutzak erabiltzea eta proiektu batean sakonean jorratzea da. Hau lortzeko, proiektu honetan, *xDrip* aplikazio irekiari funtzio berri bat gehituko zaio. *xDrip*-en integratzeko baldintzak bete behar ditu. Hala ere ezagutza lortzea da helburua, ez hedapena aplikazioan onartua izatea.

2.1.1 Helburu zehatzen deskribapena

Hauek dira **derrigorezko helburuak**:

1. *RL* erabiliz, pertsona baten *gluzemia* maila osasuntsu mantentzeko behar duen funtzioa garatzea *xDrip* aplikazioan. Funtzionalitateak pertsonak momentuan behar duen *intsulina* kantitate kalkulatu eta erakutsiko du. Kalkulu hauek *Reinforcement Learning* modelo batek egingo ditu, *gluzemia* maila balio berri bat lortzen den bakoitzean. Proiektuaren helburu nagusia aplikazioaren aldaketa denez, *modeloa*ren emaitzak ez dira lehentasun bat izango.

2. *xDrip* aplikazioan *RL modeloa* inportatzeko aukera eman behar da. Erabiltzaileak bere mugikorrean modelo fitxategia izanda, aplikazioaren bidez hau bilatu, fitxategia aukeratu eta aplikazio memorian gordeko da. Honek modeloak aldatzea ahalbidetzen du, aplikazioa aldatu gabe. Aplikazioak ez du defektuzko modelorik izango, erabiltzailearen ardura izango da hau lortzea.

3. **Kalitatezko Helburuak**: inplementazio kalitate maila minimo bat behar da. Errore kantitatea ahal bezain txikia izan behar da eta ondo probatua egon. Gainera, erabiltzailearentzako ulergarritasuna zainduko da. Inplementatutako funtzioak ezin ditu aplikazioaren beste funtzionalitateak kaltetu. Aplikazioa hedatzean funtzio berriak aurrekoen funtzionamenduarekin talka egiten ez dutela ziurtatu behar da.

4. **Irisgarritasun Helburuak:** funtzioa defektuz desaktibatuta egon behar da. Erabiltzaileak aktibatu beharko du. Aktibazioa erraz egin behar da, aplikazioaren konfigurazio atalaren bidez.

Beste aldetik, denbora izatekotan, basal *intsulina* (ikus 4.1.2. atala) gomendio sistema garatzea erabaki da. Sistema hau aukerazkoa da. Sistema honen eta derrigorrezkoaren inplementazioa printzipioz nahiko antzekoak direnez bata lortuta bestea garatzea erreza izango litzateke. Derrigorrezko sistemak bolo *intsulina* gomendioak ematen ditu. *Intsulina* hau egunean 3-4 alditan erabiltzen da gutxienez. Beste aldetik basal *intsulina*k beste helburu bat du eta beraz normalean egunero 1-2 alditan erabiltzen da.

Helburu hauek lortzeko emango diren pausuak hauek dira:

- ***OSS*-aren egoera ezagutu**: *OSS* proiektu batean lan egiteko era oso desberdina da graduko irakasgaietan edota enpresetan lan egiteko moduarekin. Hau jakinda, eremu hauek jorratu beharko lirateke proiektua ondo hasteko:
  - **Garatzaileekin komunikazioa**: *OSS*-en garatzaileak proiektuari buruzko informazio iturri handiena izan daitezke, izan ere hauek ziurrenik denbora handi bat eman dute hau garatzen. Hori dela eta, bereziki proiektu handietan, oso garrantzitsua da garatzaileekin kontaktuan jartzea. Gainera hauek proiektuan zer erabiliko den eta zer ez adieraziko dute. Beraz komunikazioa hasieratik egon behar da egin nahi den ekarpena onartua izango den jakin ahal izateko.
  - **Erabilitako erremintak**: *OSS* proiektuek kolaborazio eta bertsio kontrola erremintak erabiltzen dituzte, hala nola Git eta GitHub. Hauek nola kudeatzen eta erabiltzen jakin behar da.
  - **Proiektua ulertu**: proiektu batean aldaketak ondo egiteko proiektuaren ezagutza maila bat behar da, bereziki aldatu nahi den zatia ondo ezagutu behar da. Beraz egin nahi diren aldaketak egiteko eta hauek arazorik ez sortzeko beharrezkoa izango da aldatu nahi den eremua aztertzea eta ezagutzea. Bestalde proiektuan erabiltzen den kodetzeko era ezagutu behar da, sortutako kodea estilo berdina izan dezan.
- ***OSS* diseinu egokia sortu**: aurreko pausuak eginda, proiektua garatzeko hurrengo pausua kodearen inplementazio egokia egitea izango litzateke puntu hauek jarraituz:
  - **Hasierako inplementazio diseinua**: aplikazioaren kodea aztertzen hasita, funtzionalitate berriaren hasierako inplementazioa diseinatuko da. Hau UML bidez diseinatuko da, non garapen eta aldaketa azkarrak egin ahal diren. Inplementazio finalera hurbilen dagoen diseinua lortzen saiatu behar da.
  - **Hasierako kode inplementazioa**: hasierako inplementazio diseinua egin eta gero, hau kodean modelatzen saiatuko da. Ez da funtzionalitate finala bilatuko, funtzionalitatearen zati oinarrizkoenak inplementatzearekin nahikoa izango da. Aplikazioan funtzionalitatearen hasierako inple-

mentazio hau probatzeko aukera egon behar da, demo bat bezala lan egiteko.

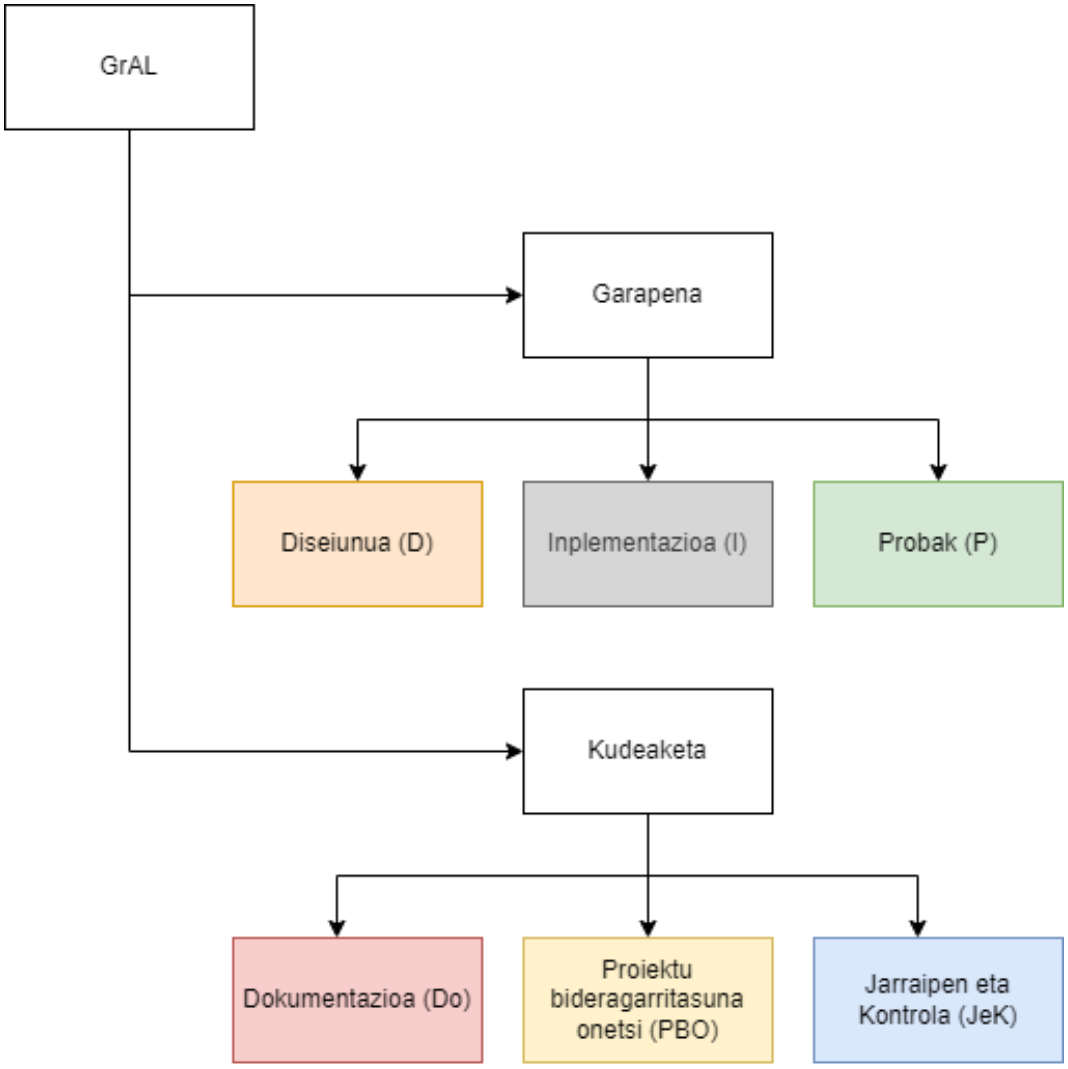
- **Kodea iteratu**: hasierako diseinua oinarri bezala hartu eta kode garapena iteratiboki garatuko da, hasierako inplementazioak gehitu behar diren funtzioen baldintza guztiak bete arte. Bukaerako iterazioan funtzioa guztiz erabiltzeko aukera egon behar da, nahiz eta gero hobekuntzak egin daitezkeen.
- **Test-ak**: aldaketak inplementatu direnean hauek ondo funtzionatzen dutela eta garatzaileentzako onargarria dela egiaztatu behar da. Ekarpenak probatu ez ezik proiektuak dituen test-ak pasatu beharko dira eta kodeari test berriak sortu. Gainera, hau lortu ondoren, garatzaileek azkeneko inplementazioa egokia den edo ez adierazi beharko dute.
- **Dokumentazioa**: garapenean zehar erabilitako orduak eta hauetan sortutako errore eta zailtasunak gordeko dira, geroago memorian hauek adierazteko eta orokorrean zer lortu den ikusarazteko.

2.1.2 Emangarriak

Lan honetan dagoen emangarri bakarra GrAL-aren memoria da eta ez dago data ezarririk hauek emateko. Hala ere 2022-2023 ikasturteko ekaineko deialdian defentsa egitea da helburua.

2.1.3 LDE diagrama

Lanaren Deskonposaketaren Egitura (LDE) diagrama aurkeztu (ikus 2.1. irudia) eta lan pakete bakoitzaren azalpen bat emango da.



Irudia 2.1: Lanaren banaketa eskematikoaren diagrama

- **Diseinua**: kodearen inplementazioaren diseinua definitzen dituen atazez osatuta dago. Diseinuak kodean inplementatuko den egitura finala islatu beharko luke.
- **Inplementazioa**: funtzionalitatearen kodearen inplementazioa lortzeko atazez osatuta, diseinua lan paketean egindako lanean oinarritzen dena.
- **Probak**: kodearen inplementazioaren proben atazez osatuta dago. Inplementazioa zuzena dela eta errorerik ez duela ziurtatu behar du.
- **Dokumentazioa**: lan-paketea GrAL-aren memoria garatzeko atazetaz osatu-

ta dago.

- **Proiektu bideragarritasuna onetsi**: proiektua denbora tarte egoki batean egin ahal dela egiaztatzeko atazak dira. Proiektua bideragarria ez dela ikusten bada proiektuaren gaia edo ikuspuntua aldatuko da.
- **Jarraipen eta kontrola**: egindako lana egokia dela eta atzerapenik ez daudela konprobatzen dituzten atazak dira.

2.1.4 Suposaketak

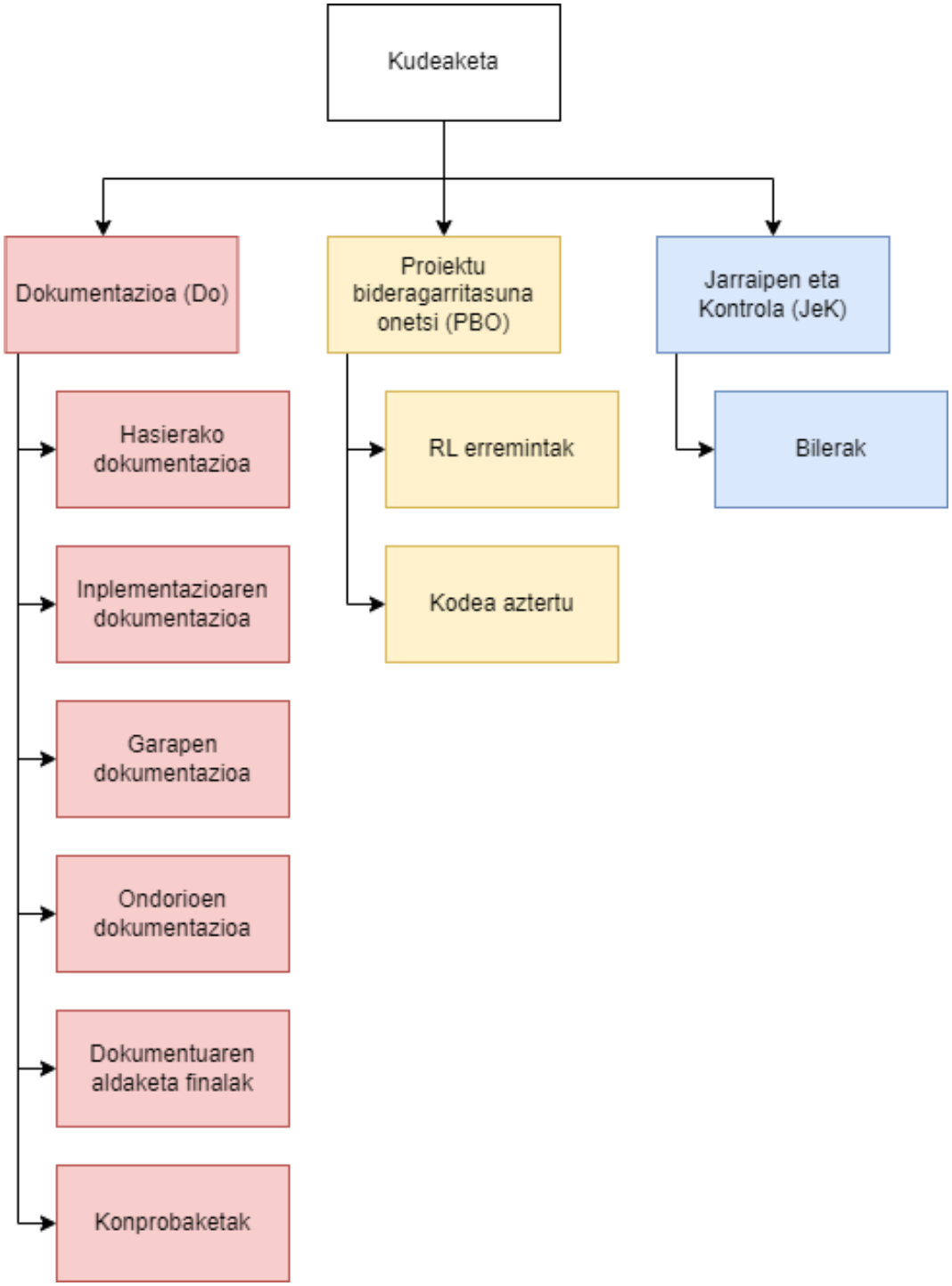
1. Garatzaileen partetik laguntza jasoko da. *OSS* proiektuak aktiboak garatzeko ekarpen bat sortzerako orduan garatzaile nagusien laguntza eta gida-lerroak nahiko lagungarri eta garrantzitsuak dira. Beraz, *OSS* askotan garatzaileen laguntza lortzea ez da oso zaila eta hauekin erraz komunikatzea posible da.
2. *RL modeloa* ez da helburu nagusia eta beraz honen emaitzei ez zaie lehentasunik emango.

2.2 Atazen denboraldiak

Lan pakete bakoitzeko ataza zehatzak adierazi eta hauen denboraldiak definituko dira.

2.2.1 Atazen deskribapena

Kudeaketa adarra
Atal honetan planifikazioaren kudeaketa atazen deskribapena egingo da (ikus 2.2. irudia). Ataza bakoitzaren ezaugarriak taula batean adieraziko dira.



Irudia 2.2: Kudeaketaren banaketa eskematikoaren diagrama

Proiektu bideragarritasuna onetsi lan paketea:

***RL* erremintak**
Lan paketea: Proiektu bideragarritasuna onetsi **Estimatutako iraupena**: 15 ordu
Deskripzioa: *RL*-an oinarritutako zati bat egin nahi dela jakinda, *modeloa* entrenatzeko simuladore bat beharko da. Hau bilatu eta erabilgarria eta egokia den konprobatu behar da.
Emaitzak/Emangarriak: Simuladorea erabiltzeko oinarrizko inplementazioa.
Baliabideak: Ordenagailua.

Taula 2.1: *RL* erremintak atazaren ezaugarri taula.

Kodea aztertu
Lan paketea: Proiektu bideragarritasuna onetsi **Estimatutako iraupena**: 15 ordu
Deskripzioa: Kodea aldatzea posible dela konprobatu eta hasierako inplementazio bat lortu. Hau egiteko gehituko diren klaseen UML diagramak sortuko dira.
Emaitzak/Emangarriak: Oinarrizko inplementazio bat eta bere UML diagrama.
Baliabideak: Ordenagailua.

Taula 2.2: Kodea aztertu atazaren ezaugarri taula.

Dokumentazioa lan paketea:

Hasierako dokumentazioa
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 20 ordu
Deskripzioa: Dokumentuaren egitura, proiektuaren helburuan azaltzen dituzten puntuak eta planifikazioa idatzi.
Emaitzak/Emangarriak: Memoria: sarrera eta planifikazioa.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.3: Hasierako dokumentazioa atazaren ezaugarri taula.

Inplementazioaren dokumentazioa
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 20 ordu
Deskripzioa: Diseinu eta kode inplementazioari buruzko informazio osoa idatzi.
Emaitzak/Emangarriak: Memoria: inplementazioa.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.4: Inplementazioaren dokumentazioa atazaren ezaugarri taula.

Beste dokumentazioak
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 9 ordu
Deskripzioa: Proiektuaren testuingurukoa den informazioa dokumentatzea, hau da, aurrekariak eta oinarri teorikoak.
Emaitzak/Emangarriak: Memoria: beste dokumentuak (aurrekariak, oinarri teorikoak).
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.5: Beste dokumentazioak atazaren ezaugarri taula.

Garapen dokumentazioa
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 20 ordu
Deskripzioa: Proiektuaren garapena azaltzen duen atala idatzi.
Emaitzak/Emangarriak: Memoria: garapena.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.6: Garapen dokumentazioa atazaren ezaugarri taula.

Ondorioen dokumentazioa
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 5 ordu
Deskripzioa: Proiektua egin eta gero garatutako ondorioen idazketa.
Emaitzak/Emangarriak: Memoria: ondorioak.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.7: Ondorioen dokumentazioa atazaren ezaugarri taula.

Dokumentuaren aldaketa finalak
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 10 ordu
Deskripzioa: Dokumentazioa egokia dela eta azalpenak era txukunean ematen direla konprobatu. Ahalik eta ortografia eta gramatika errore gutxien egotea saiatu.
Emaitzak/Emangarriak: Zuzendutako memoria.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.8: Dokumentuaren aldaketa finalak atazaren ezaugarri taula.

Konprobaketak
Lan paketea: Dokumentazioa **Estimatutako iraupena**: 14 ordu
Deskripzioa: Dokumentazioaren egoera aztertu eta hau hobetu.
Emaitzak/Emangarriak: Bukatutako memoria.
Baliabideak: Overleaf-era konektatu daitekeen ordenagailua.

Taula 2.9: Konprobaketak atazaren ezaugarri taula.

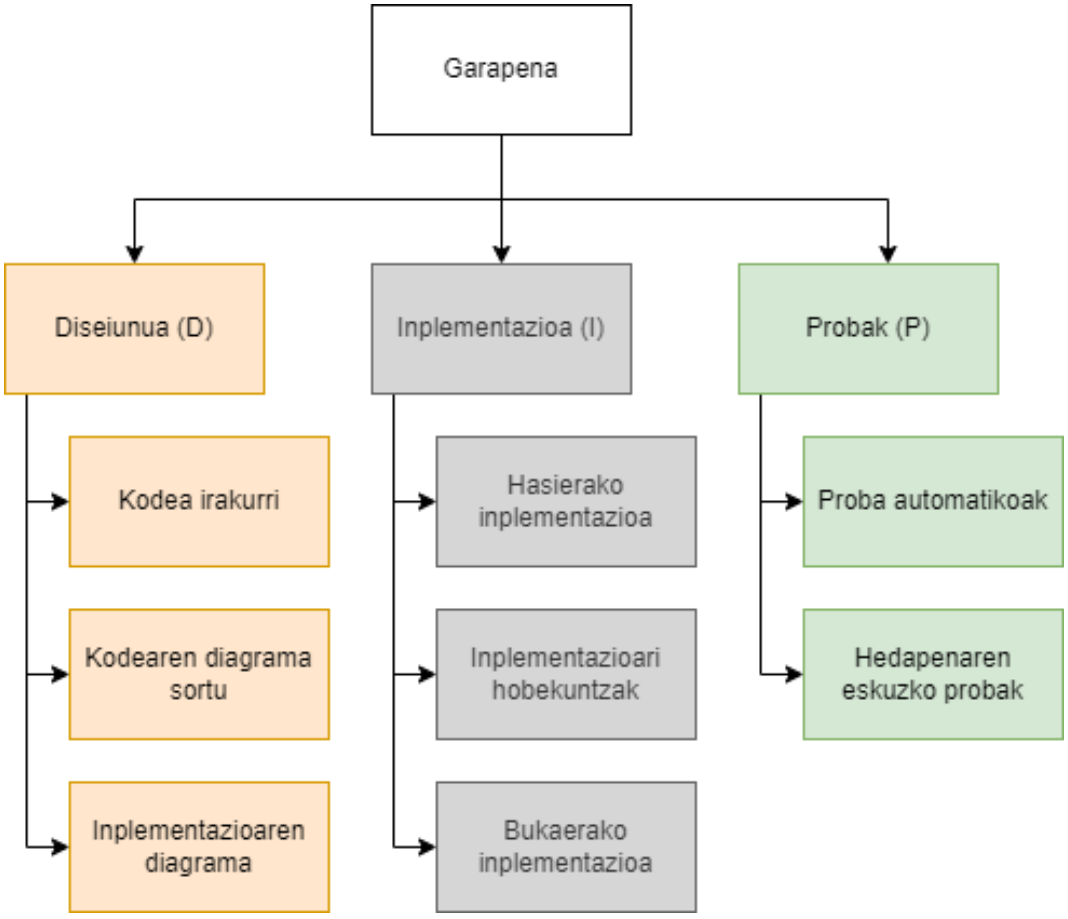
Jarraipen eta kontrola lan paketea:

Bilerak
Lan paketea: Jarraipen eta kontrola **Estimatutako iraupena**: 16 ordu
Deskripzioa: Jarraipen eta kontrola egiteko bilerak, inplementazioaren eta dokumentuaren garapenaren egoera aztertuko dira egoki doazela ziurtatzeko.
Emaitzak/Emangarriak: Bileran landutakoa eta hurrengo bileraren data eta helburuak dituen doukumentazioa.
Baliabideak: Ordenagailua.

Taula 2.10: Bilerak atazaren ezaugarri taula.

Garapen adarra

Atal honetan planifikazioaren garapen atazen deskribapena egingo da (ikus 2.3. irudia). Ataza bakoitzaren ezaugarriak taula batean adieraziko dira.



Irudia 2.3: Garapenaren banaketa eskematikoaren diagrama

Diseinua lan paketea:

Kodea irakurri
Lan paketea: Diseinua **Estimatutako iraupena**: 4 ordu
Deskripzioa: Kodearen egitura ulertzeko kodea irakurri.
Emaitzak/Emangarriak: N/A.
Baliabideak: GitHub-era konektatu daitekeen eta Java *IDE* erabili dezakeen ordenagailu bat.

Taula 2.11: Kodea irakurri atazaren ezaugarri taula.

Kodearen diagrama sortu
Lan paketea: Diseinua **Estimatutako iraupena**: 4 ordu
Deskripzioa: Proiektuaren egitura osoa ulertzeko klase diagrama sortu.
Emaitzak/Emangarriak: *xDrip* kodearen klase diagrama ulergarria.
Baliabideak: GitHub-era konektatu daitekeen eta diagramak sortzeko programa bat duen ordenagailu bat.

Taula 2.12: Kodearen diagrama sortu atazaren ezaugarri taula.

Inplementazioaren diagrama
Lan paketea: Diseinua **Estimatutako iraupena**: 4 ordu
Deskripzioa: Egin nahi den hedapenaren inplementazioaren diagrama sortuko da.
Emaitzak/Emangarriak: Hedapenaren inplementazioaren diagrama.
Baliabideak: GitHub-era konektatu daitekeen eta diagramak sortzeko programa bat duen ordenagailu bat.

Taula 2.13: Inplementazioaren diagrama atazaren ezaugarri taula.

Inplementazioa lan paketea:

Hasierako inplementazioa
Lan paketea: Inplementazioa **Estimatutako iraupena**: 75 ordu
Deskripzioa: Eskakizun guztiak betetzen dituen oinarrizko inplementazioa. Funtzionalitatea probatzeko aukera eman behar du, nahiz eta aldatu beharreko gauzak eduki.
Emaitzak/Emangarriak: Hasierako inplementazioa.
Baliabideak: GitHub-era konektatu daitekeen eta Java *IDE* bat erabili dezakeen ordenagailu bat.

Taula 2.14: Hasierako inplementazioa atazaren ezaugarri taula.

Inplementazioari hobekuntzak
Lan paketea: Inplementazioa **Estimatutako iraupena**: 75 ordu
Deskripzioa: Hasierako inplementazioa garatu, kalitatezko helburuak bete eta diseinua hobetzen saiatuko da. Kodea dokumentatuko da. Denbora izanez gero aukerazko funtzionalitate berriak garatzen hasiko da.
Emaitzak/Emangarriak: Inplementazio hobetua.
Baliabideak: GitHub-era konektatu daitekeen eta Java *IDE* erabili dezakeen ordenagailu bat.

Taula 2.15: Inplementazioari hobekuntzak atazaren ezaugarri taula.

Bukaerako inplementazioa
Lan paketea: Inplementazioa **Estimatutako iraupena**: 45 ordu
Deskripzioa: Inplementazioaren erroreak bilatu eta hauek zuzendu. Kodearen erabilgarritasuna ziurtatu.
Emaitzak/Emangarriak: Bukaerako hedapenaren inplementazioa.
Baliabideak: GitHub-era konektatu daitekeen eta Java *IDE* erabili dezakeen ordenagailu bat.

Taula 2.16: Bukaerako inplementazioa atazaren ezaugarri taula.

Probak lan paketea:

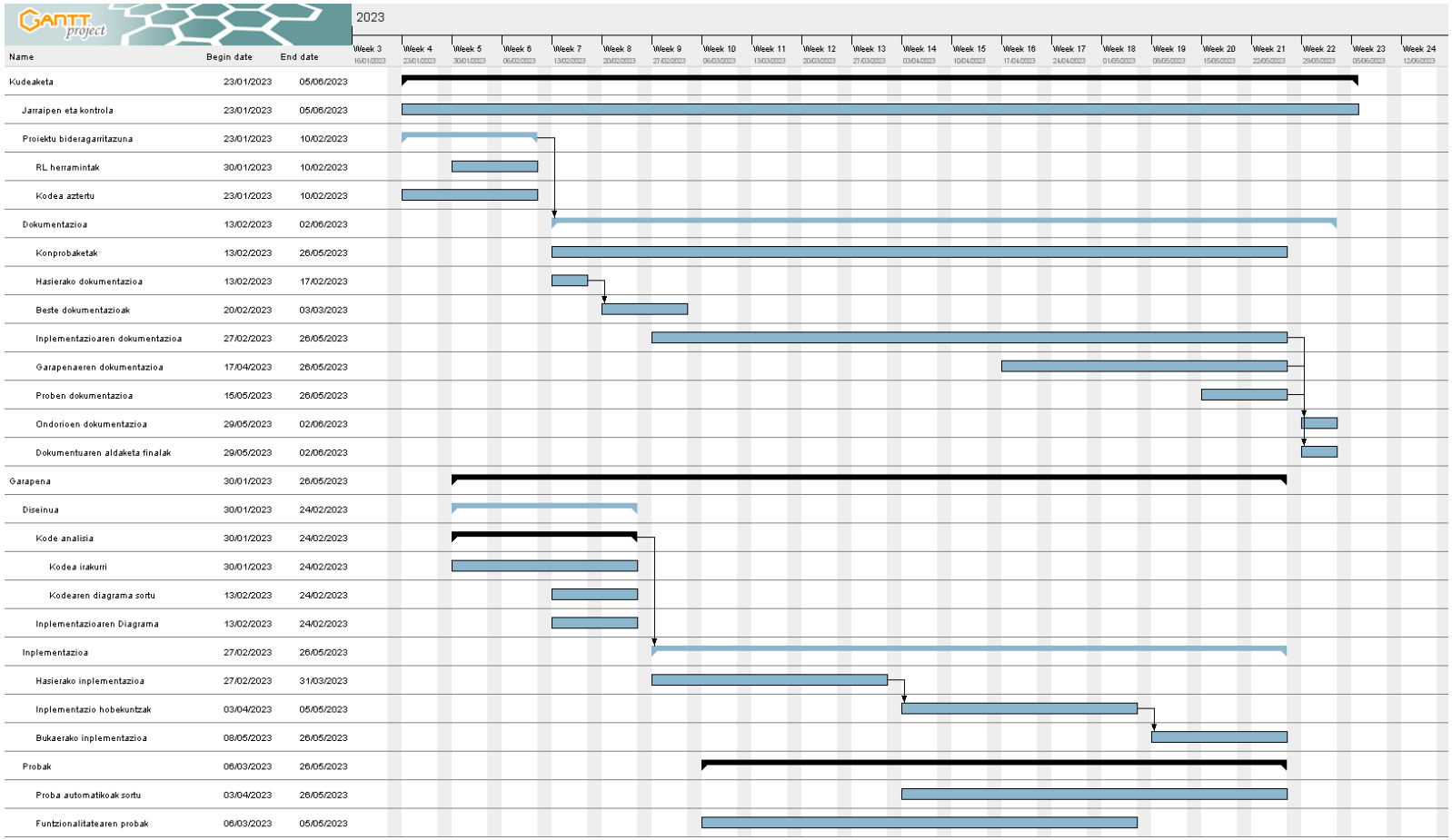
Proba automatikoak
Lan paketea: Probak **Estimatutako iraupena**: 20 ordu
Deskripzioa: Kodea automatikoki probatzen duten kodea sortu.
Emaitzak/Emangarriak: Kode probak.
Baliabideak: GitHub-era konektatu daitekeen eta Java *IDE* erabili dezakeen ordenagailu bat.

Taula 2.17: Proba automatikoak atazaren ezaugarri taula.

Hedapenaren eskuzko probak
Lan paketea: Probak **Estimatutako iraupena**: 10 ordu
Deskripzioa: Hedapena eskuz probatuko da. Proba hauek garapenean zehar egingo dira iteratiboki kode probatzeko.
Emaitzak/Emangarriak: N/A.
Baliabideak: Hedapena probatzeko ordenagailua eta mugikorra edo tableta.

Taula 2.18: Hedapenaren eskuzko probak atazaren ezaugarri taula.

2.2.2 Atazen arteko menpekotasunak eta denboraldiak



Ataza bakoitzak duen menpekotasunak eta denboraldiak adierazten dituen taula. GanttProject bidez sortu da. Hainbat ataza egiteko aurretik beste bat edo batzuk egin behar izan dira (geziek adierazten dute menpekotasuna).

2.2.3 Proiektu garapen mugarriak

GrAL-aren aurkezpenaren data ikasleak zehazten du. Beraz proiektuaren bukaera, dokumentatzeko azken eguna, 2023-ko ekainaren 5ean izatea erabaki da. Hala ere beste data mugarri batzuk sortu dira garapen egokia ziurtatzeko:

- **Hasierako inplementazioaren diagrama**. Draw.io bidez egindako diagramak sortuta egon behar dira eta kodean inplementatzeko prest. Otsailaren 24ean eginda egon behar dira.
- **Hasierako inplementazioa**. Hasierako inplementazioaren diagramaren diseinua jarraituz, beharrezko aldaketak eginda, funtzionalitatearen inplementazioa egina izatea. Martxoaren 31n egina egon beharko litzateke.
- **Inplementazio finala**. Kodearen egoera finala, probak pasatuta eta errorerik gabe. Maiatzaren 26an egina egon behar da.
- **Dokumentazioa**. Proiektuaren informazio osoa biltzen duen dokumentua eginda egotea. Ekainaren 5ean egina egon behar da, hau bidaltzeko aukerarekin.

2.3 Denbora estimazioak

2.4. irudian proiektuan lan egiteko esleitu diren denbora estimazioak agertzen dira. Ataza bakoitzeko izena, lan paketea eta estimatutako denbora agertzen da.

Kontuan hartuta planifikazioa 4. astean egin dela, 50 ordu erabili direla. 331 ordu jorratu behar dira. Ordu guzti hauek jorratzeko $331/16 = 20,625$ ordu egin behar dira astero, $20,625/5 = 4,125$ ordu egunero (kontuan hartuta larunbat eta igandetan lanik egingo ez dela). Jakinda egunero GrAL-a jorratzeko ikasleak gutxienez 4,5 ordu dituela, proiektua denboraren partetik bideragarria dela esan dezakegu.

2.4 Erremintak

Hauek izan dira erabilitako erremintak:

- Github: proiektu ireki eta itxientzako webgune eta erreminta multzoa da. Webgune honetan *xDrip* proiektua eta garapenerako erabili den *fork*-a gorde dira. Komunikazioa *xDrip* proiektuaren "Discussions" atalaren bidez egin da.
- Git: kodea eta bere bertsio desberdinak maineiatzeko erreminta da. Bertsioen arteko aldaketak egiteko erabili da.

	Denbora eztimazioak		
Kudeaketa	Proiektu bideragarritasuna onetsi	RL herramintak	15:00
		Kodea aztertu	15:00
		Guztira	30:00
	Dokumentazioa	Hasierako dokumentazioa	20:00
		Inplementazioaren dokumentazioa	20:00
		Beste dokumentazioak	09:00
		Garapen dokumentazioa	20:00
		Ondorioen dokumentazioa	05:00
		Dokumentazioaren aldaketa finalak	10:00
		Konprobaketak	14:00
		Guztira	98:00
	Jarraipen eta Kontrola	Bilerak	16:00
		Guztira	16:00
	Guztira		144:00
Garapena	Diseinua	Kodea irakurri	04:00
		Kodearen diagrama sortu	04:00
		Inplementazioaren diagrama	04:00
		Guztira	12:00
	Inplementazioa	Hasierako inplementazioa	75:00
		Inplementazioari hobekintzak	75:00
		Bukaerako inplementazioa	45:00
		Guztira	195:00
	Probak	Proba automatikoak	20:00
		Funtzionalitate probak	10:00
		Guztira	30:00
	Guztira		237:00
Proiektuaren denbora totala			381:00

Irudia 2.4: Lan pakete bakoitzeko denbora estimazioak.

- Android Studio: Google-ek sortutako eta garatutako ingurunea (*IDE*). Aplikazioa aldatzeko, konpilatzeko eta exekutatzeko erabili da.
- Jetbrains Itellij: Java aplikazioak garatzeko erabilitako *IDE*-a. Hainbat diagrama automatikoki sortzeko erabili da.
- Github Copilot: Sare Neuronalak erabiltzen dituen eta kodea sortzen duen erreminta. Sortutako kodea ez da oso ona eta oinarrizko kode egituretan laguntzeko erabili da.
- Overleaf: *LaTeX* kodea sortzeko eta maineiatzeko webgunea da. Honen bidez zuzendariek dokumentazioaren garapena ikusi ahal izan dute.
- Draw.io: diagramak sortzeko erreminta da.
- Mockito: kode probak egitean, klase batzuen portaera definitzeko erabili da.
- Robolectric: probak egitean, Android sistemaren hainbat elementu "simulatzeko" erabili da.
- Gradle: Android aplikazioak eraikitzeko dependentziak maneiatzeko eta instalatzeko erreminta.
- Simglucose: diabetesa duten gaixoen simulagailua. Diabetesa maneiatzeko algoritmoak sortzeko eta probatzeko balio du. *RL*-en oinarritutako algoritmoa garatzeko erabili da.
- *Stable Baselines 3*: *RL* algoritmoen inplementazioak eskaintzen dituen erreminta.
- OpenAI Gym: Simglucose eta *Stable Baselines 3* arteko lotura egiteko erreminta.

2.5 Kostua

Hurrengo ataletan proiektuak kostatuko lukeenaren estimazioa egingo da (ikus 2.5. irudia). Hau erabilitako denboraren eta erreminten arabera egingo da. Kontuan izanda Gradu Amaierako lan bat dela, ez da etekinik monetariorik lortuko.

Kostu laburpena	
Eskulan kostua	2.571,76 €
Softwarea	0,00 €
Hardwarea	123,00 €
Beste gastuak	480,17 €
Guztira	3.174,93 €

Irudia 2.5: Proiektuaren kostuen laburpena.

2.5.1 Eskulan kostua

Eskulan kostua kalkulatzen hasteko, zenbat ordu lan egingo diren dakigunez, orduroko kostua lortzea beharrezkoa da. *Estatuko Aldizkari Ofiziala*-ren (*EAO*)[1] arabera edozein pertsonaren soldata minimoa 1.080 euro dira. Jakinda GrAL-a garatzen duen ikasleak momentuz titulaziorik ez duela, hau oinarri bezala erabiliko da. Orain soldata orduko kalkulatu behar dugu.

$$Soldata/Orduko = \frac{Soldata\ hilabetero}{orduak\ egunero \times egunak\ astero \times asteak\ hilabetero}$$

Jakinda soldata minimoa lortzeko lanaldia egunero 8 ordu lan egitea dela, astero 5 egun, hilabete baten 4 asteetan, hau izango litzateke lortutako soldata:

$$Soldata/Orduko = \frac{1.080}{8 \times 5 \times 4} = 6.75\,€/h$$

Jakinda orduko soldata, eskulanaren kostua kalkulatuko dugu.

$$Eskulankostua = Kostua/ordu \times orudak = 6.75 \times 381 = 2.571,75\,€$$

Guztira eskulanak 2.571,75 € kostatuko du.

2.5.2 Software kostua

Proiektu osoan ez da software erremintarik ordaindu behar izan, gehienak doakoak direlako (Android Studio adibidez) edota ikasketa lizentziak erabili direlako (Github Copilot adibidez).

[1] Argitaratutako dokumentu ofiziala: https://www.boe.es/boe/dias/2023/02/15/pdfs/BOE-A-2023-3982.pdf

2.5.3 Hardware kostua

Hardwarearen partetik bi ordenagailu erabili dira, bata mahai-gainekoa eta bestea eramangarria. Beste aldetik mugikor bat erabili da aplikazioa probatzeko. Hauek dira gailu bakoitzaren amortizazioak:

$$Mahai-gainekoa = \frac{Kostua}{BalioBizitza} = \frac{1.000\,€}{10\ urte \times 12\ hilabete} = 8.33\,€$$

$$Eramangarria = \frac{Kostua}{BalioBizitza} = \frac{414.37\,€}{6\ urte \times 12\ hilabete} = 5.75\,€$$

$$Mugikorra = \frac{Kostua}{BalioBizitza} = \frac{312\,€}{4\ urte \times 12\ hilabete} = 6.5\,€$$

Beraz, jakinda gailuak 6 hilabete erabiliko direla esan daiteke $kostua\ guztira = (8.33 + 5.75 + 6.5) \times 6\ hilabete = 123$ euro izango direla.

2.5.4 Beste gastuak

Beste gastu batzuk egon dira:

- Ikasle egoitza: *GrAL*-a azken urteko 2. lauhilabetean egin denez denboraldi honetan ordaindu behar izan den egoitza kontuan hartu da. Egoitzaren kostua 523,73€ dira hilabetero. Hala ere lan egiteko 5 egun erabiliko dira bakarrik, 5 ordu egunero gutxi gora-behera. Hau kontuan hartuta

$$egoitzakostua = 523,73 \times 6\ hilabete \times \frac{5\ lan\ egun}{7\ egun} \times \frac{5\ lan\ ordu}{24\ ordu} = 467,61€$$

- Argindarra: argindarra egoitzan hilbetero ordaintzen da. Normalizatu gabe, hilabetero 14€ dira. Kontuan hartuta astean 5 egun lan egingo direla, 5 ordu egunero, $argindar\ kostua = 14 \times 6\ hilabete \times \frac{5\ lan\ egun}{7\ egun} \times \frac{5\ lan\ ordu}{24\ ordu} = 12.5€$ izango da.

Guztira, $beste\ kostuak = 467,61 + 12.5 = 480,17€$ izango dira.

2.6 Arriskuen analisia

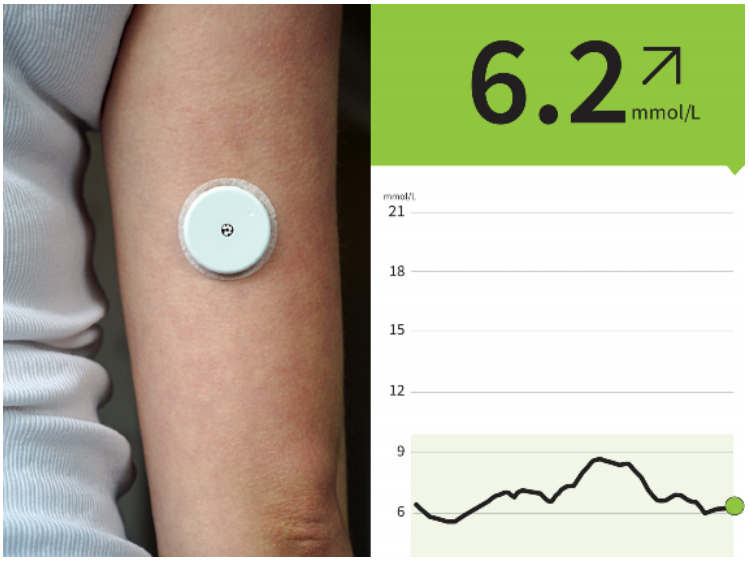
Hauek dira "Proiektu bideragarritasuna onetsi" ataza egin eta gero arrisku bezala hauteman diren atazak:

- **Erreminten bateraezintasuna**. Proiektuaren bideragarritasuna aztertzen zuen ataza egitean beharrezkoak diren *RL* erremintak bilatu eta erabili dira. Hauen arteko lorturak sortzen saiatzean hauen arteko bertsioekin arazo handiak sortu dira. Honek erreminten artean funtzionatzeko aukera ez egotea sortu dezake eta beste batzuk bilatzea edo hauek aldatzera behartuarazi.

- **Proiektuan aldaketak burutzeko zailtasunak**: *OSS* proiektu handiak ulertzeko eta aldatzeko orduan zailtasun handiak sortu daitezke. *xDrip*-en kasuan kodea oso nahastua dago eta klase kantitate altua izanda, posiblea da hainbat gauza aldatzeak aplikazio osoaren funtzionamendua zapuztea.
- ***RL* modeloak ikasteko zailtasunak**: *RL* modeloa garatzeko orduan arazoak sortu eta emaitzak onak ez izatea gertatu daiteke. Emaitzak erabilgarriak ez izatekotan, hedapenen *pull request*-a ezingo da egin, eta beraz ez da garatzaileen feedback-rik lortuko.

3 Aurrekariak

Azken urteetan diabetesaren maneiurako hainbat gailu berri sortu dira, garrantzitsuenak *CGM*-ak dira. *CGM*-ak pertsona baten *gluzemia* erraz kalkulatu dezaketen gailuak dira (ikus 4.1.3. atala). *CGM*-ek lortutako balioak ikustarazteko sistema erabiliena gaur egun mugikor aplikazioak (ikus 3.1. irudia) dira.



Irudia 3.1: Abbott *CGM*-a eta lortutako *gluzemia* balioak erakusten dituen aplikazioa.

Aplikazio hauek funtzionalitate gutxi izaten dituzte eta pertsonalizazioa murriztua da. Ondorioz, azken urteotan, erabiltzaileek diabetesaren maneiua hobetzeko proiektuak sortu eta eskuragarri utzi dituzte. Proiektu garrantzitsuenetariko bat *xDrip* da.

3.1 xDrip

2014-an Emma Black-ek DexDrip Android aplikazio irekia sortu zuen. Dexcom G4[1] aparatu medikuak ematen zituen balioak NightScout[2] aplikaziora igotzeko erabiltzen zen. Ondoren aplikazioaren izena xDrip-ra aldatu zen. Hedapen asko izan ditu urtetan zehar, hala nola gailu gehiagorekin konektatzea, datuak inportatu eta exportatu, smartwatch aplikazioa... Hedapenen ondorioz aplikazioa funtzionalitatez beterik dago, eta beraz gaur egun diabetesa maneiatzeko aplikazio ireki garrantzitsuenetariko eta erabilienetariko bat da.

Bere helburu nagusia *CGM* baten informazioa jasotzea eta erakustea da. *CGM* askorekin konektatu daiteke eta diabetesaren maneiurako funtzionalitate asko ditu. Proiektuak, GitHub orrialdean ikusi daitekeen bezala[3], eguneraketa eta mantenu handia du. Ondorioz, diabetesaren maneiuan lagundu egiten duen proiektu irekietatik erabilienetariko bat da. Funtzionalitate asko ditu, interesgarrienetariko batzuk hurrengoak dira:

1. *Gluzemia* maila aplikazioan edo smartwatch-ean ikustarazi.
2. Hitz eginez eta smartwarch bidez tratamenduak (*intsulina* injektatu, karbohidratoak jan...) biltegiratzea.
3. Erregela matematikoen bidez gluzemia aurreikusten eta *intsulina* beharra kalkulatzen saiatzen den sistema.
4. Alarmak sortzeko aukera, gluzemia ezegokia izatekotan pertsona ohartarazteko.
5. Gailu mediku askorekin bateragarritasuna dauka, hau da, euretatik datuak jaso ahal ditzake.
6. Gaixoen senideek gaixoaren datuak jaso ditzakete “companion mode” moduaren bitartez.

Diabetes gailuek erabiltzen dituzten aplikazio ofizial eta kode itxikoek funtzionalitate gutxi dituzte eta beraz *xDrip* alternatiba bat bezala erabiltzen da.

3.2 xDripen antzeko aplikazioak

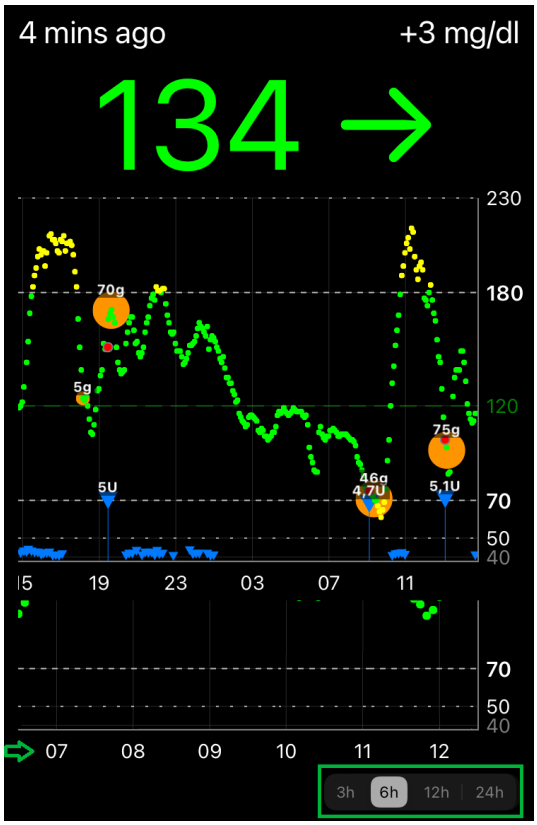
xDrip aplikazioa hedatzea erabaki da, izan ere diabetesaren maneiuan aplikazio garrantzitsuenetariko bat da. Hala ere, ez da bakarra, eta beraz helburu antzekoa duten aplikazioak eta zerbitzuak erakutsiko dira.

[1] Dexcom G4-a *CGM* bat da
[2] http://www.nightscout.info/
[3] https://github.com/NightscoutFoundation/xDrip

3.2.1 xDripSwift

xDrip Android sistema eragilean erabili daiteke bakarrik. Ondorioz *xDrip* aplikazioaren funtzionalitateak Apple-en iOS sistema eragilera ekartzen dituen xDripSwift (ikus 3.2. irudia) proiektua sortu zen. Aurretik xDrip4iOS bezala ezagutzen zen. *xDrip* ez bezala (Javaz programatuta dagoena) xDripSwift Apple-en Swift programazio hizkuntza erabiliz garatu da. *xDrip*-en bertsio mugatuago bat izanda, ez ditu Android bertsioaren funtzionalitate guztiak.

Proiektuak *xDrip*-ek baino eguneraketa gutxiago ditu. Gutxi gora-behera 1800 *commits* jaso ditu (*xDrip*-ek 7400). Gainera 30 garatzaile izan ditu *xDrip*-en 75 garatzaileekin alderatuz. GitHub orrialdean[4] ikusten den bezala, aplikazioaren dokumentazioa eskasagoa da.



Irudia 3.2: xDripSwift aplikazioaren interfaze nagusia.

[4] https://github.com/JohanDegraeve/xdripswift

3.2.2 Diabox

Bubblan[5] empresak garatutako kode itxiko aplikazioa da. Bere helburua Bubblanek saltzen duen gailuarekin komunikatzea da. Gailuak Abbott [6] enpresaren *CGM*-ek lortzen dituzten *gluzemia* balioak irakurri (ikus 3.4. irudia) eta aplikaziora bidaltzen ditu.

xDrip-ekin konparatuz, interfaze ikusgarri edo eguneratuago bat du. Beste aldetik funtzionalitate gutxiago ditu eta kode itxiko aplikazioa izateak edonork funtzionalitaterik garatzeko eta inplementatzeko aukera ixten du. Gainera gailu bat erosi beharrak (merkea ez dena) aplikazioaren erabiltzaileak potentzialak murrizten ditu.

Alabaina, gailu bat erostea eta automatikoki aplikazioarekin konektatzea *xDrip* konfiguratzea baino errazagoa izan daiteke. Beraz, erabiltzaile mota batzuentzako, gailu hauek aproposagoak izan daitezke. Hala ere, Bubblan ez da gailu mota hauek saltzen dituen enpresa bakarra, beste enpresa batzuk gailu eta aplikazio antzekoak saltzen dituzte, MiaoMiao[7] enpresa adibidez.

[5] https://www.bubblan.org/
[6] https://es.wikipedia.org/wiki/Abbott_Laboratories
[7] https://miaomiaoreader.com/a/l/es/

Irudia 3.3: Abbott FreeStyle Libre *CGM*-a, Bubblan Bubble Mini irakurlea eta Diabox aplikazioaren interfaze nagusia.

3.2.3 CamAPS FX

CamDiab enpresak eskaintzen duen aplikazioa da[8]. Aplikazioak *CGM*-en datuak jaso eta balioak ikustarazteko erabili daiteke. *xDrip*-en bezala, balioen historikoa erabiliz hainbat estatistika ikustarazten dira (batazbesteko gluzemia, gluzemia zein tartean egon den gehien,...). Aplikazioaren funtzio garrantzitsuena *intsulina* kontrol algoritmoa da. Aplikazioarekin bateragarriak diren *CGM* eta *Intsulina-Ponpa* (*IP*) konektatuta badaude, *intsulina* automatikoki injektatuko da, horretarako erabiltzaileak zenbat karbohidrato jan dituen adierazi behar du bakarrik.

Aplikazio komertziala da, harpidetza bat behar duena. Algoritmoak beraz pro-

[8] https://camdiab.com/faq

ba mediku asko jaso behar izan ditu, izan ere algoritmoak errore bat izateak ondorio larriak ekar ditzake. Ondorioz prezioa nahiko altua da, £840 urtero. Prezio oso altua dirudi baina gailu medikuen prezioak (diabetes gailuak bereziki) oso altuak izaten dira.

Irudia 3.4: CamDiab-ren CamAPS FX aplikazioa, Abbott FreeStyle Libre 3 eta MyLife Ypsopump-ekin konektatuak.

4 Oinarri teorikoak

Atal honetan diabetesaren eta *Reinforcement Learning*-en oinarri teorikoak azalduko dira. Horretaz gain, modeloa entrenatzeko erabili den simulagailuaren ezaugarri batzuk ere azalduko dira.

4.1 Diabetesa

Munduko Osasun Erakundeak adierazten duen bezala[1], diabetes sakarina edo diabetes mellitusa («diabetesa», sinplifikatzeko) gaixotasun kronikoa da. Pankreak behar adina *intsulina* sekrezioa ez duenean edo organismoak sortzen duen *intsulina* eraginkortasunez erabiltzen ez denean agertzen da.

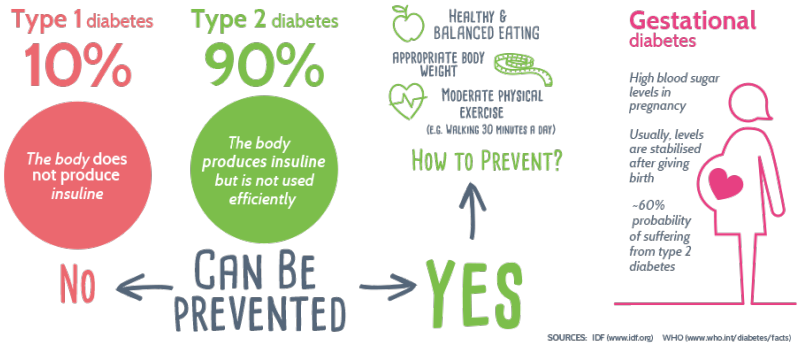
Intsulina odoleko azukre kontzentrazioa (*gluzemia*) erregulatzen duen hormona da. *Intsulina*k *gluzemia* gutxitzen du. Kontrolatu gabeko diabetesaren eragin arrunt bat hipergluzemia da (*gluzemia* altua), honek denborarekin organismoaren organo eta sistema asko larriki kaltetzen ditu, batez ere nerbioak eta odol-hodiak.

2014an, 18 urtetik gorakoen % 8,5ek diabetesa zuten. 2019an, 1,5 milioi heriotzen arrazoia izan zen, eta diabetesaren ondorioz hildako guztien % 48k 70 urte baino gutxiago zituzten. Gainera, beste 460.000 pertsona hil ziren nefropatia diabetikoaren ondorioz, eta kausa kardiobaskularren heriotzen %20 inguru hipergluzemiak eragin zituen [4].

Diabetes mota desberdinak daude (ikus 4.1. irudia), gaixotasunaren jatorriaren arabera sailkatzen dira:

- **2. klaseko diabetesa edo *intsulina* erresistentzia**: Gorputzak *intsulina* eraginkortasunez erabiltzen ez duenean gertatzen da. Gaixotasun hau garatzen laguntzen duten faktoreek pisu gehiegi izatea, ariketa fisiko eza eta genetika dira (ez da gaixotasun genetikoa, baina geneek bere garapena lagundu dezakete). Ondorioz *intsulina* injektatu edo eraginkortasuna hobetzen duten medikamenduak erabili behar dira. Diabetesa duten pertsonen % 90ek baino gehiagok 2. klaseko diabetes dute.

[1] https://www.who.int/es/news-room/fact-sheets/detail/diabetes



Irudia 4.1: Diabetes mota desberdinak.

- **1. klaseko diabetesa edo *intsulina* dependentsia**: Gorputzak *intsulina* sortzeko gaitasuna galtzen du. Gaixotasun genetikoa da, eta momentuz bere jatorria eta tratamenduak ezezagunak dira. Ondorioz, 1. klaseko diabetesa duen pertsona batek injekzioen bidez *intsulina* jaso beharko du.

Beste hainbat mota daude, diabetes gestazionala (haurdunaldian gertatzen da) adibidez, baina ez dira ohikoak.

4.1.1 Tratamendua

Diabetesaren tratamendua *intsulina*ren edo beste medikamentuen bidez egiten da. *GrAL*-ean garapena *intsulina* tratamendurako sistema batean oinarritu da, beraz hau azalduko da.

Diabetesaren tratamenduaren helburua gaixoaren *gluzemia* tarte osasuntsu batean izatea da. Munduko Osasun Erakundearen arabera[2] *gluzemia* tarte osasuntsua (barau egitean) 70-100mg/dL[3] da.

Tarte osasuntsu bat nahi bada, aldaketak sortu dezaketen aldagaiak kontuan hartu behar dira. Janariak (edo konkretuki janariak dituen karbohidratoak) *gluzemia* igotzea egingo du. *Gluzemia* jaisteko *intsulina* injektatuko da. Karbohidrato eta *intsulina* kantitate egokia aldi berean erabiltzen badira, *gluzemia* aldatzea saihestu daiteke. Beraz, karbohidratoen eta *intsulina*ren arteko ratioa ondo kalkulatzea oso garrantzitsua da, bestela *gluzemia* tartetik asko urrundu eta *gluzemia* arriskutsura heldu gaitezke:

- **Hipogluzemia**: *gluzemia* 70-etik beherago egotea da, non zenbat eta baxuago

[2] https://www.who.int/data/gho/indicator-metadata-registry/imr-details/2380

[3] mg/dL ez da *gluzemia* neurketa unitate bakarra, mmol/L erabili ohi da ere. Bien artean konbertsioa egin daiteke.

egon orduan eta sintoma larriagoak sortuko diren; zorabioak, ikusmen arazoak edo konortea galdu.

- **Hipergluzemia**: *gluzemia* 250-tik gora egotea da, gorputz zetodikoak sortuko dira, maila altuetan odoleko pH-a aldatu dezakeena. Hau luzaroan mantentzen bada organoak kaltetzen hasten dira. Organoak guztiz suntsitu daitezke.

Gluzemia arriskutsuetara ez heltzeko, injektatu behar den *intsulina*ren kalkulu egokia egin behar da. Ondorioz *gluzemia* aldatzen duten aldagaiak ezagutu behar dira. Zoritxarrez, aldagai asko daude eta gehienak ezin dira eraginkorki neurtu (estresa, beste gaixotasunik izatea,...), beraz formula sinplifikatu eta hiru aldagai hauek kontuan hartzen dira:

- **Jango diren karbohidratoak**: ia edozein janarik duen molekulak dira. "Azukre" deitu ohi zaio, nahiz eta karbohidratoak edozein janaritan ager daitezkeen, adibidez ogian. Aurretik adierazi bezala, odoleko glukosa maila igotzen dute.
- **Karbohidrato ratioa**: pertsona baten *gluzemia* berdin mantendu dadin jandako karbohidrato bakoitzeko beharrezkoa den *intsulina* kantitatea da. Hau da, karbohidrato ratioa 1:10 bada, 10 g karbohidrato jaten diren bakoitzean *intsulina* unitate bat injektatuz *gluzemia* mantenduko da.
- ***Intsulina* sentsibilitatea**: *intsulina* unitate bakoitzeko *gluzemia* zenbat unitatetan murriztuko den adierazten du. Adibidez *intsulina* unitate bakoitzeko *gluzemia* 40 mg/dL-tan murrizten bada, orduan *intsulina* sentsibilitatea 1:40 izango da.

Hiru aldagaien bidez diabetiko batek jaten duen bakoitzean edo *gluzemia* altua zuzendu behar denean, *intsulina* beharra kalkulatu daiteke. Hala ere kalkulua bere murriztapenak ditu, izan ere jaten diren karbohidratoen zenbaketa zaila edo okerra izan daiteke, eta gainera beste aldagai asko ez dira kontuan hartzen.

4.1.2 *Intsulina* motak

*Intsulina*k duen efektua (*gluzemia* jaitsi) hasieraketa denboran eta zenbat irauten duen aldatu daiteke. *Intsulina*ren arabera efektua 15 minututan edo 2 ordutan hasi daiteke eta 2 ordu edo 8 ordu tartean iraun dezake. Diabetesaren tratamenduan duten helburuaren arabera, *intsulina*k horrela sailkatzen dira[4]:

- **Bolo *intsulina***: karbohidratoak dituen janari edo edari bat kontsumitzean erabiltzen diren *intsulina*k. Kontsumitutako karbohidratoen arabera *intsulina* kantitate bat erabili beharko da. Bi motatan zatitu daitezke: efektu oso azkarrekoak, efektua 15-30 minututan hasten da eta 2-3 ordu irauten du; efektu azkarrekoak, efektua 30-60 minututan hasten da eta 5-8 ordu irauten du.

[4] https://www.diabetes.org.uk/guide-to-diabetes/managing-your-diabetes/treating-your-diabetes/insulin/types

- **Basal *intsulina***: gorputzak sortzen dituen karbohidratoek *gluzemia* ez igotzeko erabiltzen diren *intsulina*k. *Intsulina* honen bidez, barau egitean *gluzemia* berdin mantendu beharko litzateke. Bi motatan sailkatu daitezke: epe ertaineko efektua dutenak, egunero bi alditan erabiltzen dira (12 orduko efektua); efektu luzekoak, egunean behin erabiltzen direnak (24-36 orduko efektua).

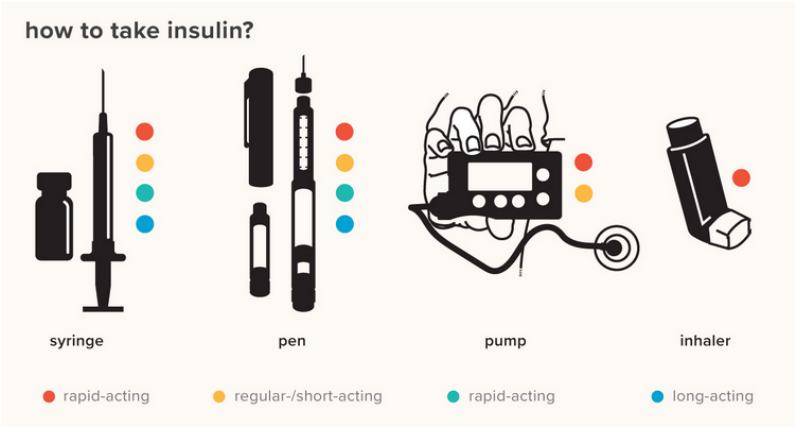
- ***Intsulina* nahastua**: bolo eta basal *intsulina* nahasten dituen *intsulina*. Janarirako erabiliko da baina bere efektua bolo *intsulina*rena baino luzeagoa izango da. Batez ere 2. klaseko diabetikoetan erabiltzen da, izan ere, 1. klasekoek bolo eta basal *intsulina* erabili behar dute.

Beste aldetik, *intsulina* hainbat eratan injektatu daiteke gorputzera (ikus 4.2. irudia):

- **Xiringak edo bolaluma**: erlatiboki merkeak izaten dira eta errazak erabiltzeko. Diabetiko gehienek sistema hau erabiltzen dute.

- **Intsulina-ponpak**: kontrol zehatzagoa ematen dute eta basal *intsulina* guztiz gelditzeko aukera du (efektu azkarreko *intsulina* erabiltzen du bakarrik). Oso garestiak dira eta arrisku batzuk dituzte, baina diabetiko baten bizitza errazten du.

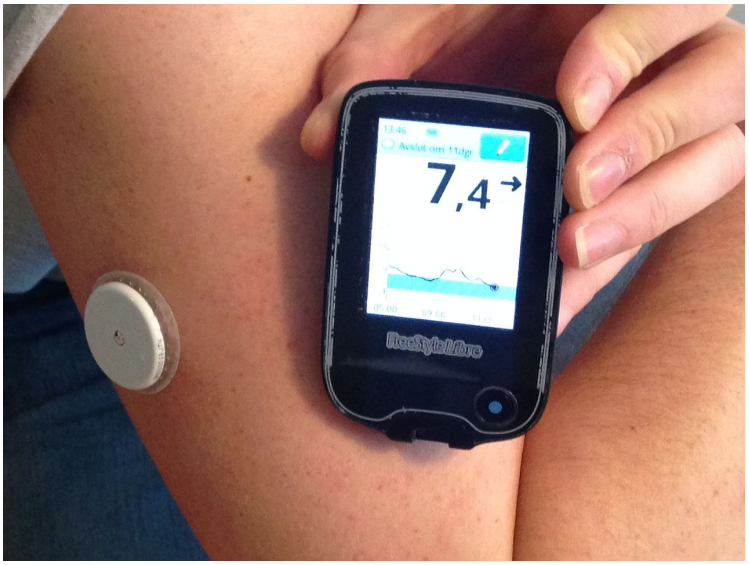
- **Inhalagailuak**: efektu azkarrenak dituzte alde handiarekin, eta ez dira mingarriak. Zoritxarrez minbizia sortu dezaketela ikusi zen eta beraz herrialde gehienetan bere salmenta debekatua dago.



Irudia 4.2: *Intsulina* gorputzean injektatzeko erak.

4.1.3 *CGM*-ak

Continuous Glucose Monitor (*CGM*) baten bidez *gluzemia* etengabe monitorizatu daiteke[5] (ikus 4.3. irudia). Gailu hauek agertu baino lehen balioak lortzeko modu errazena odol probak ziren. Odoleko *gluzemia* probak egiteko, orratz baten bidez odol pixka bat atera eta gailu baten bidez *gluzemia* kalkulatzen da (ikus 4.4. irudia). Hatzak gehiegi ez hondatzeko eta min egiten duenez, egunero odol proba gutxi egin daitezke.



Irudia 4.3: Freestyle Libre 2 *CGM*-a (besoan txertatua) eta bere irakurlearen bidez ikustarazi daitezkeen neurketak.

CGM-ek odol proben beharra ia guztiz ezabatzen dute. Gorputzean txertatu egiten dira. Behin txertatuta, plastikozko orratz baten bidez likido intertizialaren (azala eta gero dagoen geruzaren likidoa) *gluzemia* lortu dezakete. *CGM*-aren arabera maiztasun batekin egingo da kalkulua, gaur egun normalean 5 minuturo. Lortutako neurketak ikustarazteko gailu hartzaile bat beharko da. *CGM* bakoitzak hainbat egun iraun ditzake aldatu aurretik.

Gailu hartzailearen bidez neurketak jasoko dira eta beraz neurketak edozein momentutan ikusi ahalko dira. Ondorioz diabetesaren kontrol hobeagoa lor daiteke *gluzemia*ren aldaketak eta patroiak ikusi daitezkeelako.

Hala ere *CGM*-ek hainbat arazo dituzte. Odol probak baino zehaztazun txikiagoa dute. *CGM*-ak neurketa arraroak izaten baditu gomendagarria da odol pro-

[5] https://www.niddk.nih.gov/health-information/diabetes/overview/managing-diabetes/continuous-glucose-monitoring

Irudia 4.4: Odol bidez *gluzemia* lortu.

bak egitea. Gainera likido intertizialeko *gluzemia* balioak atzeratuta doaz odoleko baloiekin konparatuz (normalean 15 minutuko atzerapena), eta beraz *CGM*-en neurketak atzeratuak doaz. Gailuek duten prezioa altua izan daiteke eta beraien iraupenak hilabetean zehar hainbat erabiltzea behartzen dute.

4.2 Reinforcement Learning

Reinforcement Learning (*RL*) *Machine Learning* teknika bat da. Agente batek ingurune batean probak egitearen bidez ataza bat ikastean datza. Agenteari ez zaio ataza nola egin behar duen adierazten, ekintza bakoitzarengatik jasotzen duen sariaren arabera portaera garatuko da.

RL-a ataza sekuentzialetarako erabiltzen da. Agenteak ingurunean ekintza bat egitean hurrengo egoerara helduko da eta beste ekintza bat hartu beharko du. Ekintzaren bidez egoera hobeago bat lortu dela jakiteko, inguruneak agenteari sari bat emango dio. Agenteak helburura hurbiltzeko behar den ekintza bat hartu badu sari bat jasoko du, eta zigortua izango da helburutik urrundu bada.

Beste *ML* teknikek, ikasketa gainbegiratuak eta ez-gainbegiratuak, ez dituzte sariak erabiltzen. Ikasketa gainbegiratuaren entrenamenduan (ikasketak) datu eta lortu nahi den emaitza bikoteak erabiltzen dira ataza baten funtziora hurbiltzeko. Beste aldetik ikasketa ez-gainbegiratuak datuak bakarrik erabiltzen ditu eta datu hauen patroiak bilatzen ditu.

Badira beste desberdintasun batzuk ere. *RL*-en, beste ikasketa metodoetan ez

bezala, denbora garrantzitsua da. Ikasketa gaibegiratu eta ez-gainbegiratuek datu bat jaso, emaitza kalkulatu eta prozesua bukatzen dute. Ez dira iraganeko edo etorkizuneko datuak kontuan hartzen. Beste aldetik *RL*-n ez dago gainbegiraketarik, ingurunean lortzen den saria da emaitza gidatuko duen gauza bakarra.

4.2.1 *RL*-ren elementuak

RL-ren helburua agente batek ingurune batean hartu behar dituen ekintza hoberenak ikastea da. Hau egiteko hurrengo elementuak erabiltzen dira. Elementu bakoitza definitzerako orduan labirinto baten adibidea erabiliko da, non helburua labirintotik ateratzea izango den ahalik eta pausu gutxien emanaz.

1. **Agentea**: ataza bat egiten ikasten duten sistema. *RL* bidez ikasi duen sistema sare neuronala bat izan daiteke adibidez. Adibidean, agentea labirintotik atera nahi den izakia izango litzateke.

2. **Ingurunea**: ikasi nahi den ataza deskribatzen du. Adibidean, atazaren ingurunea labirintoa izango litzateke.

3. **Egoerak**: agenteak ingurunean duen egoeraren deskribapena. Adibidean, agenteak labirintoan duen posizioa izango litzateke.

4. **Ekintzak**: agenteak ingurunean har ditzakeen erabakiak dira. Labirinto adibidearen kasuan agenteak aurrera, atzera, ezkerrera edo eskuinera mugitzea izango litzateke. Egoeraren arabera ekintza batzuk ezinezkoak izango dira, adibidez agenteak bere aurrean pareta badu, ezingo da aurrea joan.

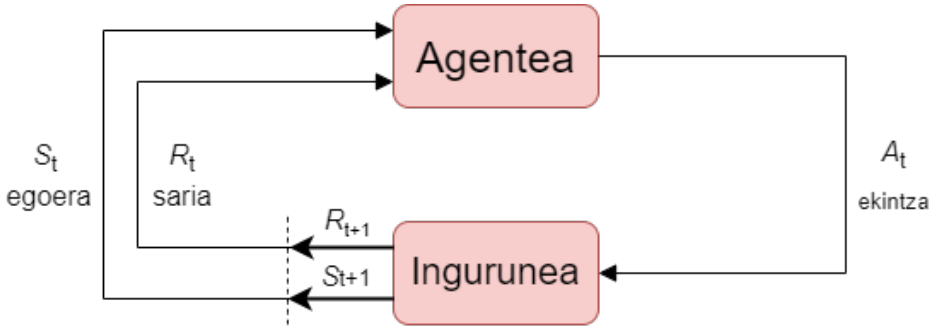
5. **Sariak**: agenteak ekintza bat egitean inguruneak ematen dion feedback-a. Erabaki den ekintzak egoera hobeago edo txarrago bat lortu duen desberdintzen du. Labirintoaren kasuan, saria labirintotik ateratzerakoan eman daiteke, saria handiagoa izango da geroz eta pausu gutxiagotan ateratzea lortzen bada.

4.2.2 Oinarrizko *RL* ikasketa zikloa

RL-ren ikasketa 4.5. irudian erakusten da. Atazaren t denbora-urrats bakoitzeko, agenteak ingurunean duen $S_t \in S$ egoera jasoko du (S egoera posible guztien multzoa izanda). S_t arabera agenteak $A_t \in A(S_t)$ ekintza aukeratuko du ($A(S_t)$ uneko egoeran aukeratu daitezkeen ekintza posible guztiak izanda). Ekintza bat aukeratua, $t+1$ denbora-urrantsera eta S_{t+1} egoerara mugituko da agentea, $R_{t+1} \in \mathbb{R}$ sari bat jasoz. $R_t + 1$ sariak A_t ekintza aukeratzearen onura adierazko du.

4.2.3 Markov-en Erabaki Prozesuak

RL-en atazak matematikoki definitu daitezke *Markov Decision Process*-ren (*MDP*) bidez. *MDP*-a $M = \langle S, A, T, R \rangle$ tupla osagaiez osatua egongo da:



Irudia 4.5: Oinarrizko *RL* ikasketa zikloa.

1. S: egoera multzo mugatua.

2. A: ekintza posible guztien multzo mugatua. Hala ere ekintzak egoeren araberakoak direnez uneko ekintza $A_t \in A(S_t)$ bezala definitzen da, egoera bakoitzeko ekintza multzo konkretu bat egongo baita.

3. T: egoeren arteko trantsizio probabilitateak definitzen ditu. Ataza determinista (ikus 4.6. irudia) batean bagaude $T : S \times A \rightarrow S$ egoera baten trantsizioa berdina izango da ekintza berdina hartzen den bakoitzean (labirintoan adibidez). Beste aldetik ataza estokastiko (ikus 4.6. irudia) batean $T : S \times A \times S \rightarrow P(S)$ egoera batean ekintza batek ausaz hainbat egoera desberdinetara trantsizionatu dezake (dado jokoak adibidez). Beraz $T(s'|s,a)$ a ekintza erabiliz s egoeratik s'-ra aldatzeko probabilitatea definitzen du. Matematikoki horrela definitu daiteke:

$$T(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\}$$

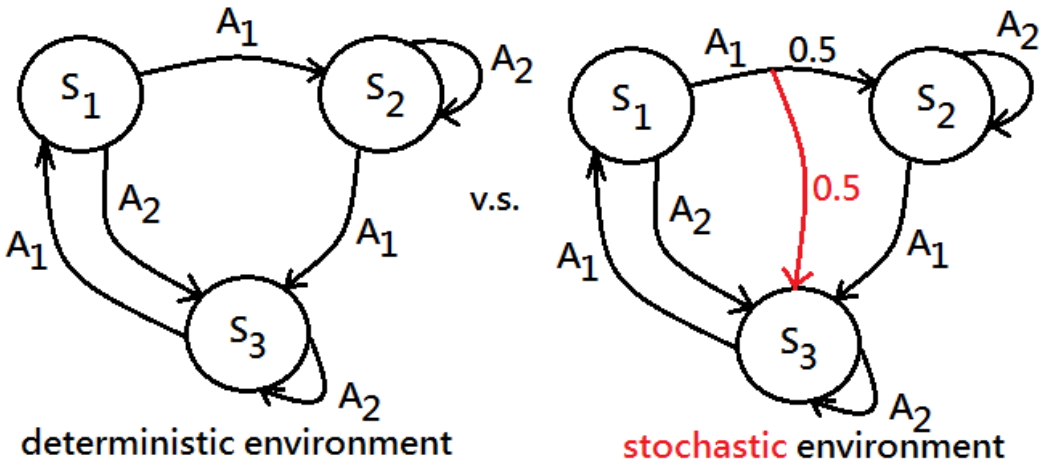
   T probabilitate banaketa bat definitzen duenez, hurrengo betetzen da:

$$\sum_{s' \in S} T(s'|s,a) = 1, \forall s \in S, \forall a \in A \tag{4.1}$$

4. R sari funtzioa da. $s \in S$ egoeratik hasita, $a \in A$ ekintza hartuz eta $s' \in S$ egoerara heltzean lortzen den r saria definituko du. r sariak egoera berrian egoteko onura adieraziko duen zenbaki erreala da.

MDP-ek gainera Markov-en propietatea betetzen dute. Markov-en propietatearen arabera: S_{t+1} egoerara heltzeko probabilitatea uneko egoerararen (s) eta hartutako ekintzaren (a) araberakoa izango da bakarrik. Hau da, ingurunearen aldaketa uneko egoeren eta ekintzen araberakoa da, iragana ez da kontuan hartzen. Uneko egoerak ekintza bat aukeratzeko informazio guztia izango du. Matematikoki horrla definitzen da:

$$\begin{aligned} &Pr\{S_{t+1} = s', R_{t+1} = r|S_t, A_t\} = \\ &Pr\{S_{t+1} = s', R_{t+1} = r|S_t, A_t, R_t, S_{t-1}, A_{t-1}, \ldots, R_1, S_0, A_0\} \end{aligned} \tag{4.2}$$



Irudia 4.6: *MDP* determinista eta estokastikoa.

4.2.4 Helburuak eta sariak

Agenteak egoera berri batera heltzean sari bat jasoko du. Agenteak saria maximizatzen saiatuko da. Hala ere egoera aldaketa saria ez ezik, denboran zehar lortutako sari kopuru guztia kontuan hartu behar da, hau da, G_t epe luzerako saria edo itxarondako saria.

$$G_t = R_{t+1} + R_{t+2} + \dots R_T, \quad T < \infty \tag{4.3}$$

Epe luzerako sariak baldintzatzen duen bi ataza mota daude, amaiera duten eta jarraituak diren atazak. Lehenengoek hasiera eta amaiera dute, amaieran bukaerako egoerara helduz. Beste aldetik ataza jarraituek hasiera dute baina amaierarik ez, eta beraz epe luzerako sariaren 4.3 formulak ez du ondo funtzionatzen $T = \infty, G_t = \infty$. Hau zuzentzeko $0 \leq \gamma \leq 1$ deskontu faktorea erabiliko da, sarien balioa gero eta zaharragoa izan gero eta balio gutxiago izan dezaten:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.4}$$

γ balioaren arabera saria epe luze edo laburrekoa den definituko du. $\gamma = 0$ bada uneko saria soilik kontuan hartuko du (epe laburrekoa), eta $\gamma = 1$ bada sari guztiak kontuan hartuko ditu (epe luzekoa).

4.2.5 Politikak

Agenteak G_t itxarondako saria maximizatu behar du. Beraz agenteak portaera zehatz bat garatu beharko du egoeraren arabera hartuko duen ekintza definitzeko.

Portaera politika (π) bezala definitzen da. Politikak deterministak $a = \pi(s)$ (egoeraren arabera beti ekintza onena aukeratu) edo estokastikoak $\pi(a|s) = Pr\{A_t = a|S_t = s\}$ (egoeraren arabera aukeratuko den ekintza probabilitate banaketa bat jarraituz aukeratu) izan daitezke.

Agentearen politikak bere portaera definituko du eta beraz *RL*-aren helburu nagusietako bat politika hoberena aurkitzea da. Politika hoberena, π^* bezala definitzen da, politika posible guztien artean itxarondako sari batura handiena duena izango da.

4.2.6 Balio funtzioa

RL bidezko ikaskuntza algoritmo asko balio-funtzioen hurbilketaren bidez lortzen saiatzen dira. Balio funtzio erabilienak egoera-balioak (V) eta ekintza-balioak (Q) dira.

$V^\pi(s)$ egoera-balio funtzioak s egoeran egotean eta π politika jarraituz itxarongo den epe luzerako saria definitzen du.

$$V^\pi(s) = \mathbb{E}\pi[G_t|S_t = s] = \mathbb{E}\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{4.5}$$

non $\mathbb{E}\pi$ espero den G_t definitzen duen.

$Q^\pi(s, a)$ ekintza-balio funtzioak s egoeran egotean, a ekintza hartzen eta π politika jarraituz itxarongo den epe luzerako saria definitzen du.

$$Q^\pi(s) = \mathbb{E}\pi[G_t|S_t = s, A_t = a] = \mathbb{E}\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{4.6}$$

4.2.7 Bellman-en ekuazioak

Egoera eta ekintza balioak uneko sariaren eta hurrengo sari deskontatuaren gehiketa bezala definitu daitezke. Ekuazio hauek Bellman-en formula bezala ezagutzen dira. Egoera-balioarentzako Bellman ekuazioa hurrengoa da:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots |S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots)|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_(t+1) + \gamma V^\pi(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} T(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')] \end{aligned} \tag{4.7}$$

Beste aldetik ekintza-balioaren Bellman ekuazioaren definizioa hurrengoa da:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t|S_t=s,A_t=a] \\ &= \mathbb{E}_{\pi}[R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|S_t=s,A_t=a] \\ &= \mathbb{E}_{\pi}[R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\ldots)|S_t=s,A_t=a] \\ &= \mathbb{E}_{\pi}[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a] \\ &= \mathbb{E}_{\pi}[R_{t+1}+\gamma Q^{\pi}(S_{t+1},A_{t+1})|S_t=s,A_t=a] \\ &= \sum_{s'} T(s'|s,a)[R(s,a,s')+\gamma(\sum_{a'}\pi(a'|s')Q^{\pi}(s',a')]] \end{aligned} \tag{4.8}$$

4.2.8 *RL* ebazpen metodoak

RL ikasketa metodoak bi tekniketan banatu daitezke:

1. **Tauletan oinarritutako metodoa**: egoera edo ekintzen tamaina handiegia ez denean erabiltzen dira. Taulen metodoen erabilera bat 1000 lauki dituen labirinto bat ebaztea izango litzateke. Taulen metodoak *Markov Decision Process* (*MDP*) bidez definitzen dira eta hauek bi motatan banatu daitezke:

   (a) **Modeloan oinarrituta**: agenteak ingurunea guztiz ezagutzen du, hau da, ingurunearen informazio guztia ezagutzen da.

   (b) **Modelogabea**: agenteak ingurunearen ezagutza partziala du, sari distribuzioa eta/edo trantsizio probabilitatea ezezagunak dira.

   Hainbat taula metodo daude, hauetatik garrantzitsuenak Programazio Dinamikoa (modeloan oinarritua), Monte Carlo (modelogabea) eta Denborazko Desberdintasun (modelogabea) metodoak.

2. **Funtzio hurbilpeneko metodoak**: egoera edo ekintza asko edo infinitu dauzkagunean erabiltzen direnak. Metodo hauetan ohikoena sare neuronal artifizialak erabiltzea da. Proiektu honetan sortu den *intsulina* gomendio sistema garatzeko mota honetako metodoak erabili dira.

4.3 Simuladorea

xDrip-en erabiliko den *RL modeloa* garatzeko diabetesaren simulagailu bat behar da. Simulagailuak 1. klaseko diabetiko (ikusi 4.1. atala) bat simulatu behar du, hau da, kanpoko *intsulina* bere kabuz sortzeko gaitasuna ez duen eta injekzioak jaso behar dituen pertsona. Simulagailua eta *RL* algoritmo bat erabiliz, proba eta errore bidez, *intsulina* maneiatzen ikasiko duen *RL modeloa* sortuko da.

Man et al.-ek [5] proposatutako simulagailua aukeratu da. Simulagailu honen Python inplementazioa "Simglucose"[6] (*SG*) izenarekin ezagutzen da. Simulagailua nahiko eguneratua dago (2014-koa) eta "Simglucose" eguneraketak jasotzen

ditu oraindik. Ikerketetan erabiltzeko Estatu Batuetako Janari eta Medikamentuen Administrazioaren onarpena du. Gainera simulagailua erabili duten hainbat ikerketa aipagarri daude, haien artean: Lee et al. [7], Zhu et al. [2], Hettiarachchi et al. [8], Yau et al. [9],... Beste aldetik *Simglucose*-ek *OpenAI Gym* (*Gym*) *API*-arekin lotura du inplementatuta.

Gym lotura erabiliz *Stable Baselines 3* (*SB3*) bezalako *RL* liburutegiak erabili daitezke. *SB3* bidez *intsulina* maneiatuko duen agentea sortuko da. Agenteak simulagailuan eragingo du eta emaitzak aztertuko ditu.

Simulagailuaren *Gym* loturak ekintza bakarra baimentzen du, *intsulina* injektatzea. Agenteak *intsulina* kantitatea aukeratu eta ekintzaren emaitza jasoko du. Emaitza bezala behaketa (*gluzemia* neurketa), *RL* saria, bukaera egoera (simulazioa bukatu ote den adierazten duen boolearra) eta informazio gehigarria (*RL* ikasketan erabiliko ez den baina garapenean ondo datorren informazioa) jasoko da.

Simglucose-k agenteen ikasketan laguntzeko hainbat funtzionalitate ditu, hala nola hainbat diabetiko mota egotea. Diabetesaren eragina gorputzean adinaren arabera asko aldatzen da, eta beraz haur, nerabe eta helduak simulatzen ditu. Adin mota bakoitzeko 10 gaixo desberdin daude, bakoitza desberdintasun batzuekin. Gaixo bakoitza simulazio berezitu bat denez, simulagailuak hainbat gaixo paraleloan simulatzeko gaitasuna du.

Simulagailuak beste ezaugarri erabilgarriak ditu: gaixoek karbohidratoak jatea, agentearen ekintzak eta beren emaitzak grafiko bidez ikustaraztea, aurretik inplementatutako oinarrizko kontrolagailuak,... Agente bat garatzeko beharrezkoak diren ezaugarri gehienak ditu eta beraz *intsulina* maneiatzen duten agenteen hainbat ikerketetan erabili da.

5 Betekizunen espezifikazioa

5.1 Sarrera

Atal honetan *xDrip* aplikazioari gehitu nahi zaizkion funtzionalitateen betebeharren espezifikazioa zehaztuko da. Funtzionalitateak aukeratzeko irizpideak hauek izan dira:

- 5 hilabetetan, 300 ordu inguru sartuz, egin ahal den funtzionalitatea.
- Proiektuaren interfazea, datu sistema eta hauek konektatzen dituen negozio logika aldatzeko beharra duen funtzionalitatea.
- Garatuko den funtzionalitatea dagoeneko aplikazioak duen funtzionalitate baten antzekoa izatea, kodea nola egituratu behar den aztertzeko.

Reinforcement Learning (*RL*) modelo baten bidez *intsulina*ren ziztatze-kontrol sistema adimentsu bat egitea aukeratu da. Sistema honek pertsona baten *gluzemia* ("azukrea" odolean) maila jakinda, beharrezkoa duen *intsulina* kantitate bat gomendatuko du. Bi *intsulina* mota daude, bolo *intsulina* (efektu azkarra, 3 ordukoa) edo basal (efektu motela, egun osokoa) *intsulina*. Desberdin lan egiten dutenez, bakoitzeko funtzionalitate bat inplementatu daiteke.

Lehenengoz bolo *intsulina* sistema garatzea erabaki da, erabiliko diren erreminten ondorioz errezena delako. Basal *intsulina* sistema ez da garatu denbora faltagatik.

5.2 Betekizun funtzionalak

Bolo ziztatze-kontrol adimentsuaren betekizun funtzionalak ondorengoak dira:

- Erabiltzaileak funtzionalitatea aktibatzeko eta desaktibatzeko aukera izango du.
- Erabiltzaileak *RL modeloa*ren fitxategia inportatzeko aukera izango du.

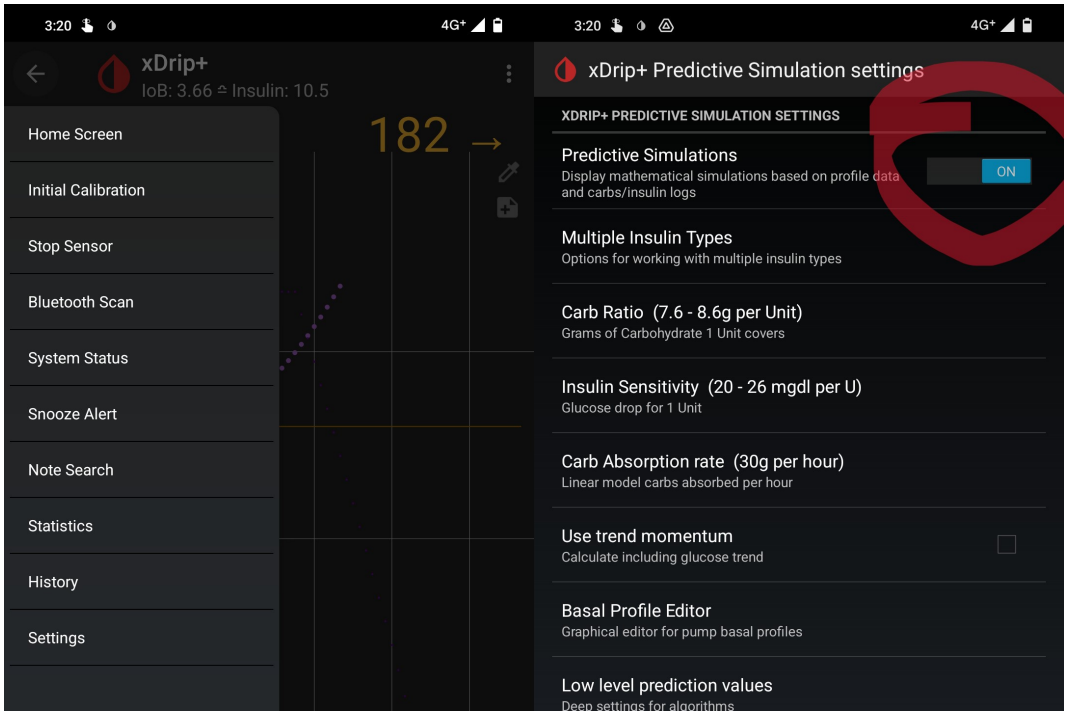
- Erabiltzaileak, funtzioa aktibatuta egotean, *intsulina* ziztatze-kontrol adimentsuak gomendatzen duen *intsulina* ikus dezake.
- Modeloa zuzena ez bada (erroreak izan edo ondo definitua ez egon) ez da *intsulina* gomendiorik adieraziko.

5.3 Hedapenen ezaugarriak

Atal honetan funtzioen erabilgarritasuna, fidagarritasuna, errendimendua eta bateragarritasuna azalduko dira.

5.3.1 Erabilgarritasuna

Funtzionalitatea aktibatzea erreza izan behar da. Horretarako, aplikazioaren konfigurazio atalean, aktibazio botoi bat (ikus 5.1. irudia) egongo da. Hala ere funtzioa esperimentala da, eta hori adieraztea beharrezkoa izango da.



Irudia 5.1: Nola aktibatu funtzioa

5.3.2 Fidagarritasuna

Funtzioak zuzen lan egin behar du, bat-bateko errorerik eman gabe. Edozein errore egotekotan, aplikazioak erabiltzailea notifikatu behar du. Konkretuki *RL mo-*

*deloa*k errorerik edukitzeak *GrAL*-ean garatu den funtzionalitatea bakarrik ezgaituko du (ez dira gomendiorik kalkulatuko), eta errorea erabiltzaileari argi adieraziko zaio.

5.3.3 Errendimendua

Aplikazioaren errendimendua ez da kaltetu behar. *RL* modelo bat erabiliko da. *RL* modeloa aukeratzean, arina izatea gomendatzen da. Hau da, erabiltzaileak potentzia handia eskatzen duen modeloa kargatzeak errendimendua kaltetu lezake. Erabaki hau erabiltzailearen eskuetan egongo da.

5.3.4 Bateragarritasuna

Funtzionalitateak erabiltzen duen modeloa edozein momentutan aldatu daiteke (hedapenaren modelo inportatze botoiarekin), berdin definitutako modelo desberdin batekin. Honen bidez beste edozein pertsonak modeloa sortu eta aplikazioan probatu dezake koderik aldatu gabe. Modelo berriek ezaugarri konkretu batzuk izan behar dituzte (sarrera eta irteera konkretuak adibidez). Gida sinple bat sortu behar da modeloa nola definitu eta erabili behar den adierazteko.

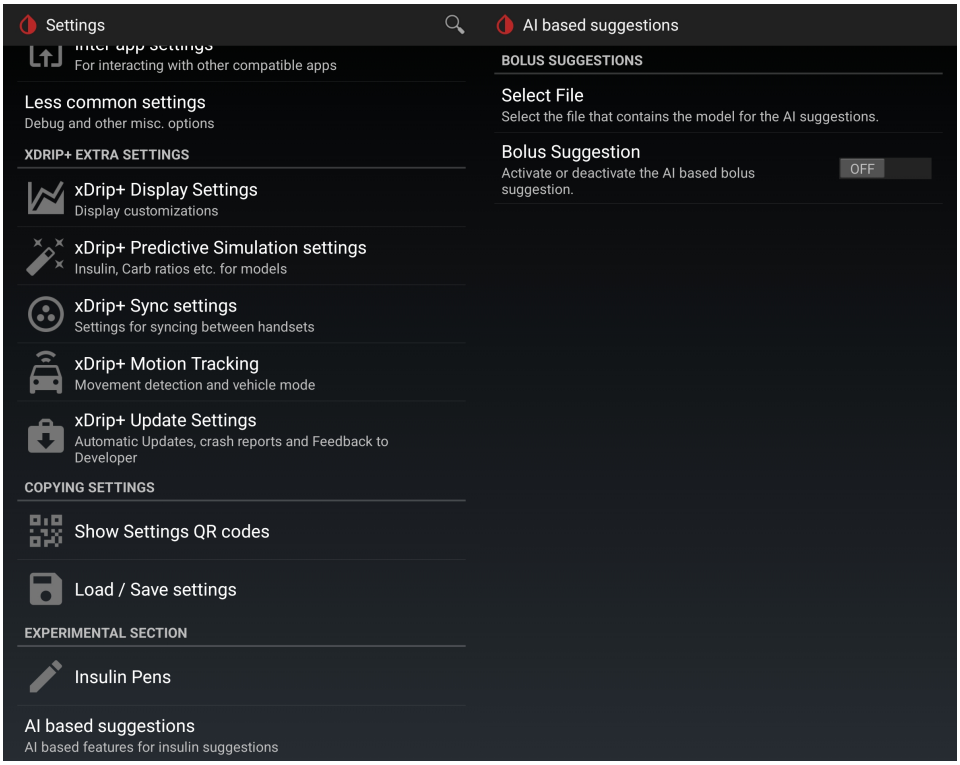
5.4 Interfazeak

Hauek dira sortu beharreko interfazeak:

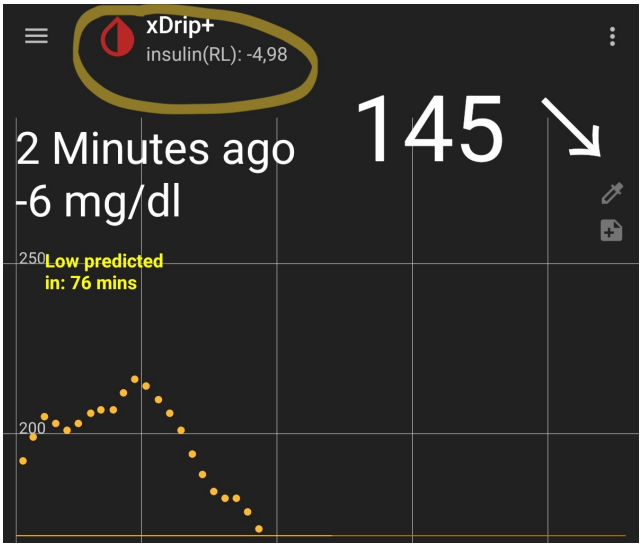
- Konfigurazioaren eremuan, atal esperimentalean, "AI based suggestions" atal berri bat gehitu (ikus 5.2. irudia). Funtzionalitate bakoitzak sekzio bat izango du, bere izenburuarekin eta hurrengo elementuekin:
  - **Aktibazio botoia**: funtzionalitatea aktibatzen eta desaktibatzen duen botoia (edo "silder"-a).
  - **Modeloa inportatzeko botoia/atala**: sakatzean, modeloaren fitxategia (mugikorraren memorian) non dagoen adierazteko botoia.
- Uneko *gluzemia* erabiliz, sistema adimentsuak gomendatzen duen *intsulina* kantitatea adierazten duen interfazea (ikus 5.3. irudia). Sistemak errorerik izatekotan "error" mezua adieraziko du.

Itxura eta informazioa

Interfaze berriek aplikazioaren beste interfazeen itxura/estilo berdina izan beharko dute.



Irudia 5.2: "AI based suggestions" eremua eta bolo sistema adimentsuaren interfazea.



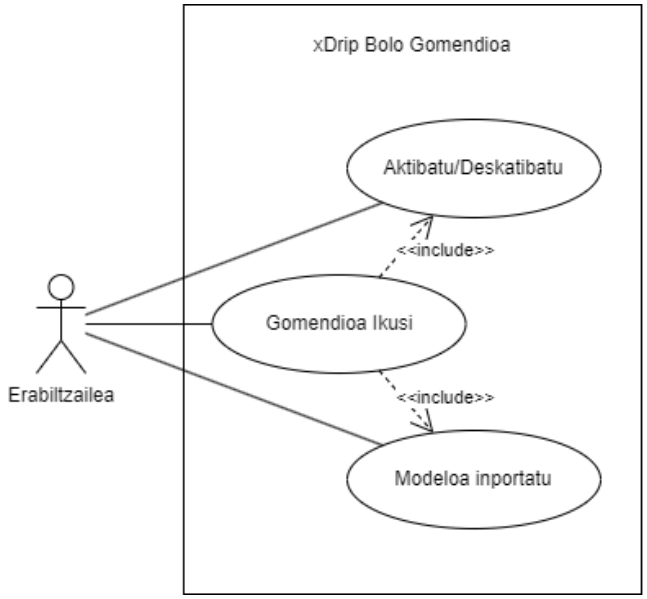
Irudia 5.3: Emaitza erakusteko interfazea

5.5 Erabilpen kasuak

Atal honetan (bolo) *intsulina* gomendio sistemaren erabilpen kasua azalduko da (5.4. irudia erabilpen diagrama).

5.5.1 Intsulina gomendio sistema

Erabiltzaileak, funtzioa aktibatu eta *RL* modeloaren (*intsulina*ren gomendioa kalkulatuko duen sistema) fitxategi zuzena inportatuko du. Modeloa ondo kargatzen bada, pertsonak momentuan behar duen *intsulina* gomendioa kalkulatuko da. Kalkulu honek *xDrip*-ek duen formula matematikoan oinarritutako funtzionalitateak baino hobeto lan egin beharko luke (formula matematikoak ez ditu *gluzemia* patroiak kontuan hartzen).



Irudia 5.4: Bolo *intsulina* gomendio sistemaren erabilpen diagrama.

5.6 Mugarriak

- RL modelo on bat lortzeko baliabide eta denbora gehiago beharko litzateke. Beraz ez dira emaitza onak edo erabilgarriak eskatuko modeloarentzako.
- Erabiliko den simulagailuak muga asko ditu. Nahiz eta ikerketarako sortua eta erabilia izan den, mundu errealeko eta simulagailuaren emaitzak ez dira berdinak izango. Gainera simulagailua era batera lan egiteko prestatuta dago

(sarrera eta irteera konkretu batzuk adibidez) eta aldaketak egitea neketsua izan daiteke.

- *xDrip* aplikazioan hedatuko da eta beraz kode berria aplikazioaren kodera moldatu behar da. Gainera, kodea proiektu nagusira hedatua izatea nahi bada, garatzaileen oniritzia jaso behar da.

5.7 Betekizunak legalak

5.7.1 Lizentzien betekizuna

xDrip aplikazioaren GitHub web-gunean adierazten den bezala[1], lizentzia irekiko euskarriak erabili ahal dira bakarrik.

5.7.2 Ohar legala

xDrip aplikazioan egindako lan guztia proiektuaren lizentziapean (GPL) egon behar da, eta beraz, Unai Hurtadoren lana izan arren, lizentzia horretan adierazten diren arauak mantendu eta mugak onartu behar dira.

5.8 Dokumentazioa

Garapen dokumentazio guztia memoria honetan egongo da. https://www.overleaf.com/project/

[1] https://github.com/NightscoutFoundation/xDrip/blob/master/CONTRIBUTING.md

6 Analisi eta diseinua

Atal honetan aplikazioaren funtzionalitate berriak inplementatzeko analisia eta diseinua azalduko da. Analisi atalean *xDrip* aplikazioaren kodearen eta *RL modeloa* garatzeko azterketak azalduko dira. Diseinu atalean *xDrip* aplikazioari gehitu behar izan zaizkion klaseen azalpena emango da.

6.1 Analisia

Proiektuan bi analisi desberdindu dira. Hasteko bideragarritasun analisia egin da. Ondoren, proiektua bideragarria dela ziurtatu eta gero, *xDrip* aplikazioan aldaketak egiteko analisia azalduko da. Atal honetan klase eta sekuentzia diagramak erabiliko dira aplikazioaren egoeraren azalpena errazteko.

6.1.1 Bideragarritasunaren analisia

xDrip-en kodea ez ezagutzeak eragindako ziurgabetasunaren ondorioz eta *RL modeloa* sortzeko baliabideak definitu beharra zegoenez, hedapenen egingarritasuna bideraezina izateko aukera zegoen (kodea nahasgarria izatea, modeloarentzako tresnarik ez egotea...). Bideragarritasuna ziurtatzeko *GrAL*-ren lehenengo bi aste erabili dira. Analisia bi zatitan banatu da, *xDrip* aplikazioaren analisia (bi ataletan zatituta, *xDrip*-en analisi orokorra eta *xDrip*-en kode analisia) eta *RL modeloa* garapena. Hauek biak aztertu eta gero, proiektua bideragarria dela ondorioztatu da.

xDrip-en analisi orokorra

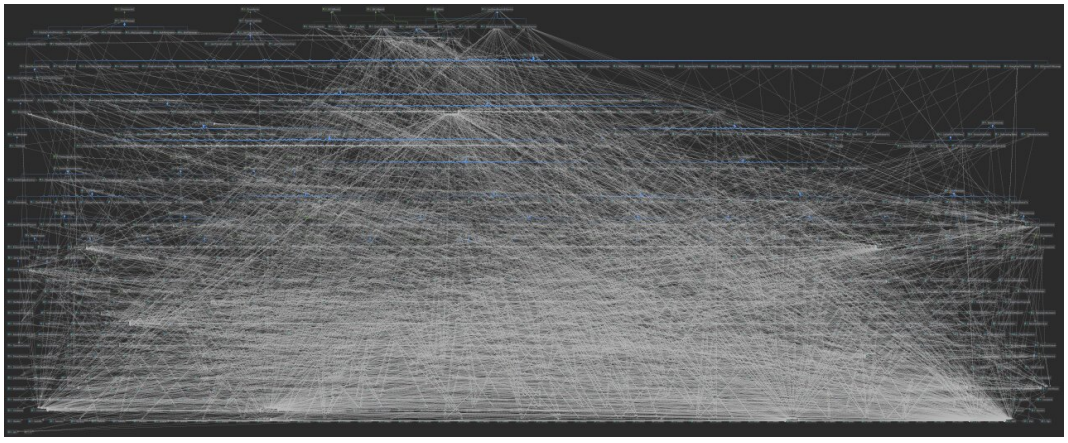
xDrip-en funtzionalitate berriak gehitzeko bideragarritasuna aztertu da. Horretarako, aplikazioaren egitura aztertu, aldatuko diren klaseak bilatu eta klaseen portaeran ulertu da. **Egitura aztertzeko** hurrengo pausuak eman dira:

1. **Kodearen dokumentazioa bilatu**: *Open Source Software (OSS)* handienetan, normalean kodearen dokumentazioa eskura dago. Hau ez da *xDrip*-en kasua. Ez dira metodoen edo kodearen egituraren dokumentazio idatzirik edo diagramarik aurkitu. Aplikazioaren erabileraren dokumentazioa[1] du baina

[1] https://xdrip.readthedocs.io/en/latest/

ez kodearen antolaketari buruzkoa.

2. **Proiektuaren diagrama sortu**: kodearen egitura deskribatzen zuen dokumentaziorik ez aurkitzean, proiektuaren klase diagrama sortzea erabaki da. Jakinda klase diagrama guztiak eskuz egiteko denbora gehiegi behar dela, proiektuaren klase diagramak automatikoki sortzen dituen programa bat bilatu da. JetBrains Intellij *IDE*-ak emaitza hoberena eman du, baina ezin da kodearen azterketa bat egiteko erabili (loturak eta klase kantitatea handiegia da). 6.1. irudian proiektuak diten Java klaseak (900 baino gehiago) eta dependentziak ikusi daitezke.



Irudia 6.1: *xDrip*-eko Java klase guztien arteko dependentzia diagrama Intellij bidez sortua.

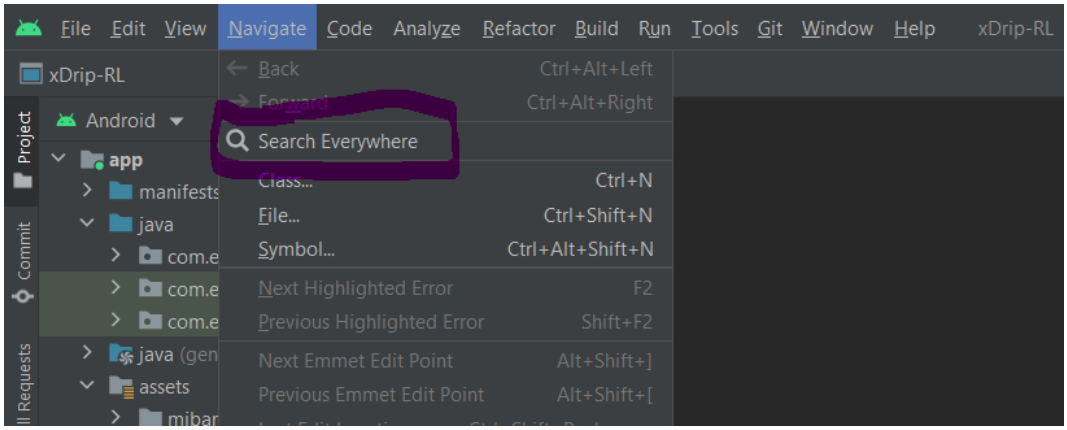
3. **Kodea aztertu**: kodearen dokumentaziorik aurkitu ez denez eta klase diagrama automatikoki sortzean lortutako emaitza erabilgarria ez denez proiektuaren kodea zuzenean aztertu da, Android Studio *IDE*-a erabiliz. Kodea aztertzen zen bitartean proiektuaren egituraren eta kodearen klase diagramak sortu dira.

xDrip-en kodearen analisia

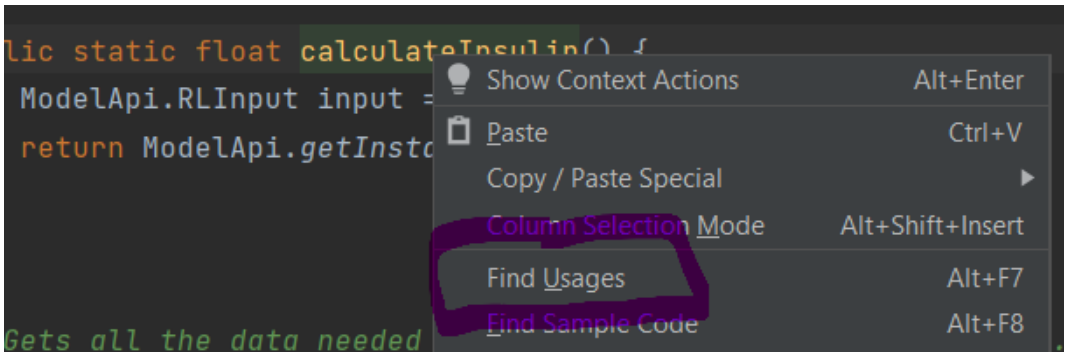
Kodearen analisia egiteko Android Studio-k dituen erremintak erabili dira. Gehien erabili diren erremintak "search everywhere" (ikus 6.2. irudia) eta "find usages" (ikus 6.3. irudia) dira.

Hasteko, aplikazioaren diseinu orokorra aztertu da. Aplikazioa hiru Java moduluetan banatuta dago:

- **App**: *xDrip* Android aplikazioaren Java modulua da. Aldatu nahi diren interfazeak eta kodea ditu, beraz sakonean aztertu behar da. Denbora eskaini



Irudia 6.2: Android Studio-k duen kodea testu bidez bilatzeko erreminta: "search everywhere".

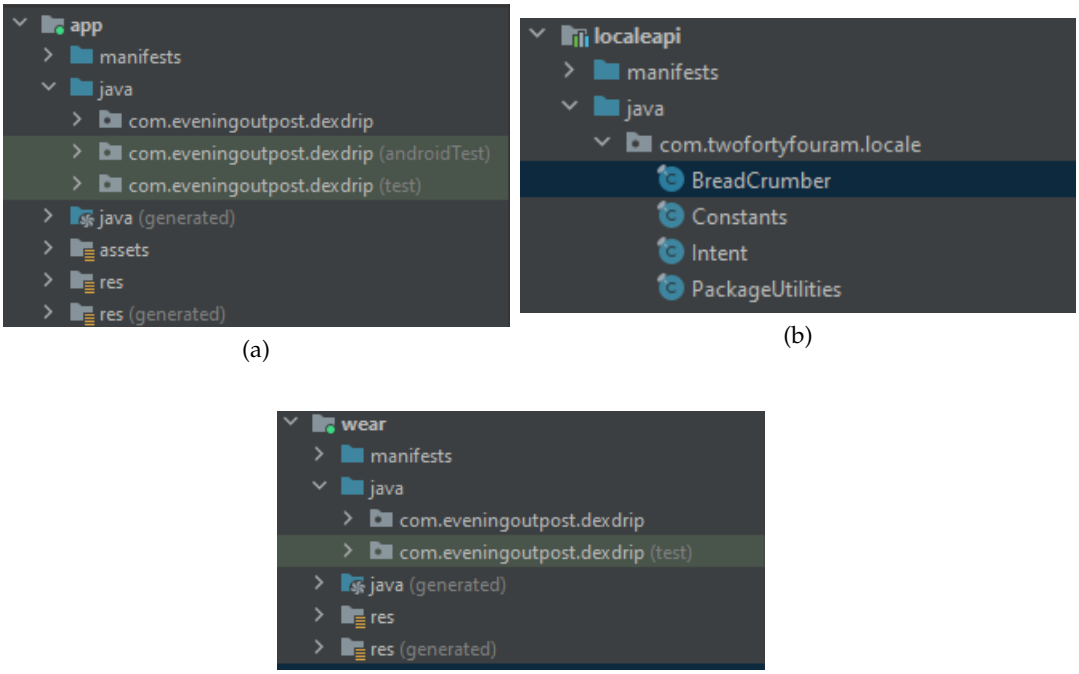


Irudia 6.3: Android Studio-n klase bat non erabili den adierazteko erreminta: "find usages".

behar zaio aztertzeari, modulu honek dituen Java klase kantitateren, klaseen izen eta ordenaren eta bakoitzak duen dependentzien (ikus 6.1. irudia) ondorioz.

- **LocaleApi**: modulu honen helburua ez da oso argia. Hala ere hedatu nahi diren aplikazioaren eremuak hemendik at daudenez eta hauek aldatzen ez direnez (horretarako klase bakoitza zein eremutan erabiltzen den begiratu da), ez da sakonago aztertu.
- **Wear**: *xDrip*-en WearOS[2] aplikazioaren Java modulua. Smartwatch-entzako aplikazioaren modulua da. Android aplikazioa bakarrik aldatu nahi denez, ez da sakonean aztertu.

[2]Google-n smartwach sistema operatiboa.



(a) (b)

(c)

Irudia 6.4: Java moduluak.

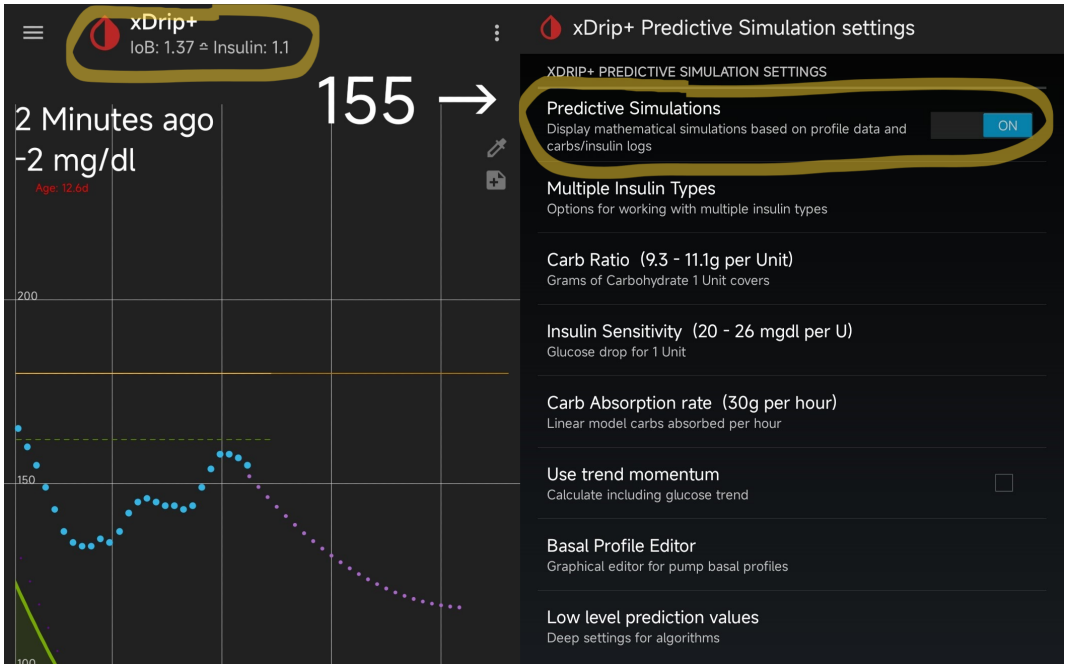
Moduluen egitura jakinda, "app" moduluan sakondu eta aldatu behar diren klaseak bilatuko dira. Zorionez, *xDrip*-en garatzaile nagusiak datuak lortzeko erabili daitezkeen Java klaseak zeintzuk diren adierazi dizkigu (aurretik sortutako *discussion*-ean[3]). Beste aldetik, eskuz bilatu behar izan dira aplikazioan aldatu behar diren interfazearen eta interfazeekin lan egiten duen kodea.

Funtzionalitate berrien interfazeak sortzeko *xDrip*-ek duen antzeko (formula matematikoetan oinarritutako) funtzionalitatearen inplementazioa oinarri bezala hartu da. Funtzionalitate honek garatu nahi denaren portaera ia berdina du (*glu-zemia* balio berri bakoitzeko *intsulina* beharra kalkulatzen du, emaitza interfazean ikustarazten du, aktibatu/desaktibatu daiteke...)

Android aplikazioek normalean xml bidez zehazten dituzte interfazeak. Hauek "res/xml" fitxategian gordetzen dira. Hau jakinda, 6.5. irudian ikusten den konfigurazio interfazea hasiera puntu bezala hartu da, kodetik klase diagrama lortzeko (alderantzizko ingeniaritza). "find usages" eta "search everywhere" erabili dira horretarako.

Hainbat konfigurazio fitxategi daude aplikazioan (ikus 6.6 irudia). Kodea azter-

[3] https://github.com/NightscoutFoundation/xDrip/discussions/2651



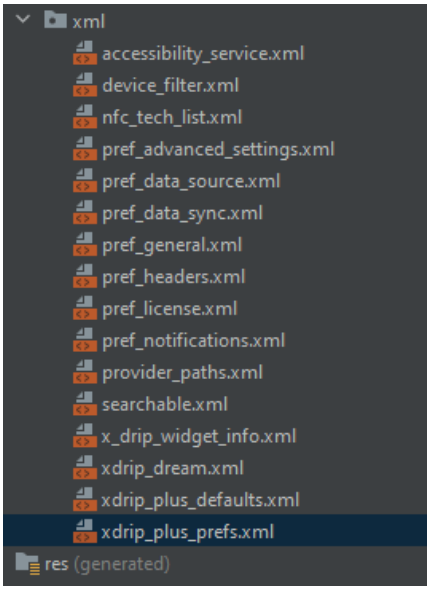
Irudia 6.5: Aplikazioaren interfaze nagusia eta konfigurazio interfazea.

tu eta gero argi geratzen da "xdrip_plus_prefs.xml" fitxategiak oinarri bezala hartu nahi den konfigurazioa duela eta fitxategi hau aldatu beharko dela (ikus 6.7 irudia).

"xdrip_plus_prefs"-en formula matematikoetan oinarritutako funtzionalitatearen aktibatze botoia aztertu da. Funtzionalitatea aktibatzeko xml-ren "key"-a (edo "simulations_enabled" deitzen den xml-an definitzen den aldagai globala) hartu eta "search everywhere" bidez kodean non erabiltzen den bilatu daiteke.

"Key" hau "BgGraphBuilder" klasean erabiltzen da. Klase honen bidez funtzionalitatean erabiltzen den kodea aztertu da. Klasearen "addBgReadingValues" (817 lerroko luzera du) metodoa erabiliz 6.5. irudian ikusten den grafikoa sortzeko eta goian adierazten den "Insulin:" string-a gehitzeko metodoa deitzen du. Hau aldatzean "Insulin:" string-a aldatzeko eta beste edozer gauza ipintzeko era aurkituta (eta hau *gluzemia* lortzen den bakoitzean eguneratzen denez), aplikazioaren kodearen aldaketa bideragarria dela ondorioztatu da.

Datuak lortzeko klaseak aztertzea errazagoa izan da. Garatzaile nagusiak gomendatu zuen bezala "Treatments" eta "BgReadings" klase estatikoak aztertu dira. Biek "latest(x)" metodoa dute, azken tratamenduak (*intsulina*, janaria...) eta *gluzemia* balioak lortzen dituztenak. Beraz, ez dago arazorik datu hauek lortzeko.



Irudia 6.6: *xDrip*-en "App" moduluaren interfazearen xml-ak.

```
<PreferenceScreen
    android:dependency="I_understand"
    android:icon="@drawable/ic_auto_fix_grey600_48dp"
    android:key="xdrip_plus_profile_settings"
    android:summary="Insulin, Carb ratios etc. for models"
    android:title="xDrip+ Predictive Simulation settings">

    <PreferenceCategory
        android:key="xdrip_plus_profile_category"
        android:title="xDrip+ Predictive Simulation settings">
        <SwitchPreference
            android:defaultValue="true"
            android:key="simulations_enabled"
            android:summary="@string/display_mathamatical_simulations"
            android:switchTextOff="OFF"
            android:switchTextOn="ON"
            android:title="Predictive Simulations" />
```

Irudia 6.7: "xdrip_plus_prefs.xml" fitxategia. *xDrip*-en gomendio sistemaren interfazearen zatia ikusgarri.

***RL modeloa*ren garapena**

Gradu Amarierako Lana (GrAL)-ren oinarria *xDrip* hedatzea izan arren, funtzio berria erabiltzeko *RL* modeloa beharrezkoa da. Modeloak egoera bat deskribatzen dituen datuak jasota, egoera horretan ekintza hoberena zein den kalkulatuko du (*GrAL*-aren kasuan zenbat *intsulina* behar den). *RL* modelo bat sortzeko bi osagai nagusi behar dira, ingurunea eta algoritmoa. *GrAL*-aren kasuan modeloaren ingurunea diabetesaren simulagailu bat da. Beste aldetik ingurune honetan ikasiko duen algoritmoa sortu eta entrenatu behar da.

Hasteko diabetesaren hainbat simulagailu bilatu dira: Aida [10], "Minimal Model: Perspective from 2005" [11], "Qualitative behavior of a family of delay-differential models of the glucose-insulin system." [12]. Arrakastatsuenetariko bat "The UVA/-PADOVA Type 1 Diabetes Simulator" [5] (Padova) da. Padova algoritmoaren inplementazio ireki bat sortu zen Python erabiliz, "simglucose" [6] deitzen dena. Hainbat lan zientifikoetan erabili da [7, 3, 13, 14].

Beste aldetik, modeloa sortzeko algoritmoa behar da. *GrAL*-ren oinarria aplikazio hedapena denez, modeloaren garapena ahal bezain beste sinpletu da. Hau lortzeko Stable Baselines 3 (SB3)[4] erabili da (*RL* garatzeko liburutegia). *RL* hainbat algoritmoren inplementazioak ditu eta inplementatzerako orduan erraztasunak ematen ditu.

Simglucose ingurunea eta *SB3* izanda, biak lotzeko sistema behar da. Bi sistemek OpenAI Gym-ekin (*Gym*) lan egiteko prestatuta daude, beraz hau erabiliz hasierako inplementazio sinple bat sortu da proiektuaren bideragarritasuna aztertzeko.

6.1.2 Aplikazioaren analisia

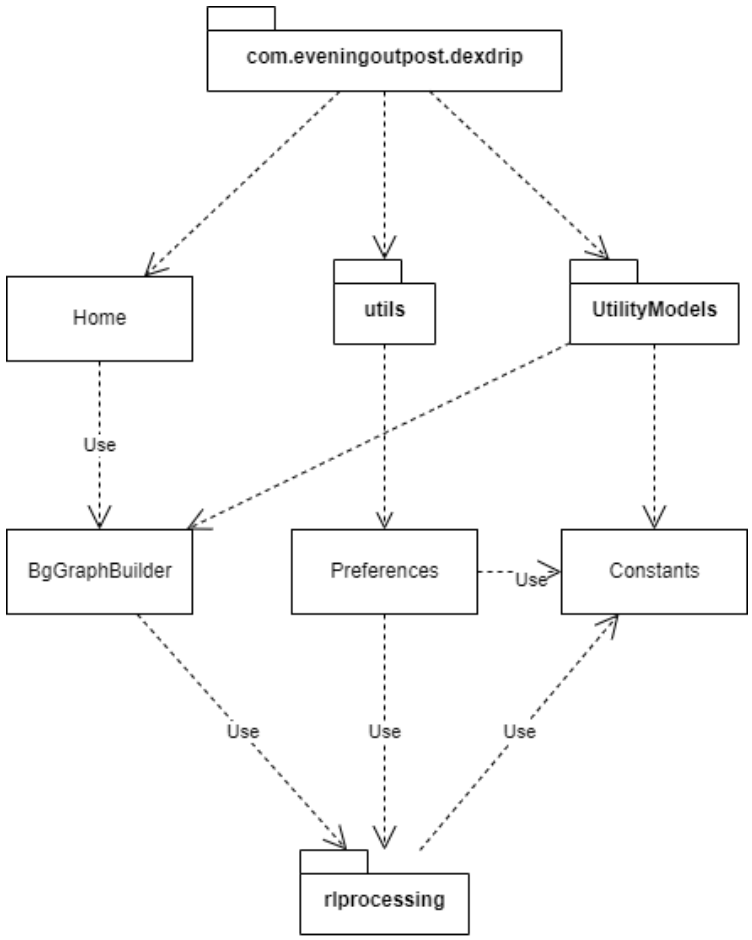
Bideragarritasuna baieztatu eta gero, funtzionalitatea inplementatzeko kodearen azterketa egin da. Analisian klase hauek aldatu behar direla identifikatu da (ikus 6.8. irudia):

- **Constanst**: datu prozesaketa klaseetan sortutako konstanteak gordetzeko.
- **Preferences**: konfigurazio interfazea aldatzen den bakoitzean (funtzio bat aktibatu edo desaktibatu) aldaketak jasoko eta prozesatuko ditu. *RL modeloa*ren fitxategia inportatzeko funtzioak beharko duen "handler"-a sortzeko ere hau aldatu beharko da.
- **BgGraphBuilder**: aplikazioaren interfaze nagusiaren grafikoa sortzeko klasea da. Grafikoa interfaze nagusira sartzean sortu eta *gluzemia* balio bat ja-

[4] https://stable-baselines3.readthedocs.io/en/master/

sotzean eguneratzen da. Grafikoak eguneratzean *intsulina*ren beharraren kalkuluak egiten dira, gero hauek "Home" klaseak jasotzeko.

- **Home**: aplikazioaren interfaze (ikus 6.5. irudiaren ezkerreko interfazea) nagusia maneiatzen du. Android-en erabiltzen diren broadcast sistemaren bidez *intsulina* beharren eguneraketak jasotzen ditu.



Irudia 6.8: *xDrip* aplikazioan aldatu beharko diren klaseen diagrama.

Orain aldatu behar izan diren klaseen kodearen azalpena egingo da. Bi zatitan banatu da: *RL* modeloa inportatzeko kodearen analisia eta *intsulina* gomendioa ikustarazteko kodearen analisia.

RL modeloa inportatzeko kodearen analisia

5.4. atalean adierazten den bezala, modeloa inportatzeko interfazeko botoi baten bidez egingo da. Beraz konfigurazioaren interfazea maneiatzen duen klasea

("handler" edo kontrolatzailea) aldatu behar da. Klase hau eskuz bilatu da, eta hainbat klase aztertu eta baztertu ondoren, "Preferences" klasea aztertu da. Klase honek, "PreferenceActivity" klasea hedatzen du. "PreferenceActivity" klasea Android klasea da, konfigurazio egitura erakusteko eta maneiatzeko erabiltzen dena[5].

"Preferences" klasearen barruan "AllPrefsFragment" klasea dago gordeta. "AllPrefsFragment" klasea aztertzean konfigurazio interfazean erabiltzen diren hainbat ezaugarriren kontrolatzaileak ("handler") aurkitu ahal dira. Beraz, esan daiteke modelo inportatze sistema "AllPrefsFragment" klasea aldatuz inplementatu daitekeela.

Intsulina gomendioa ikustarazteko kodearen analisia

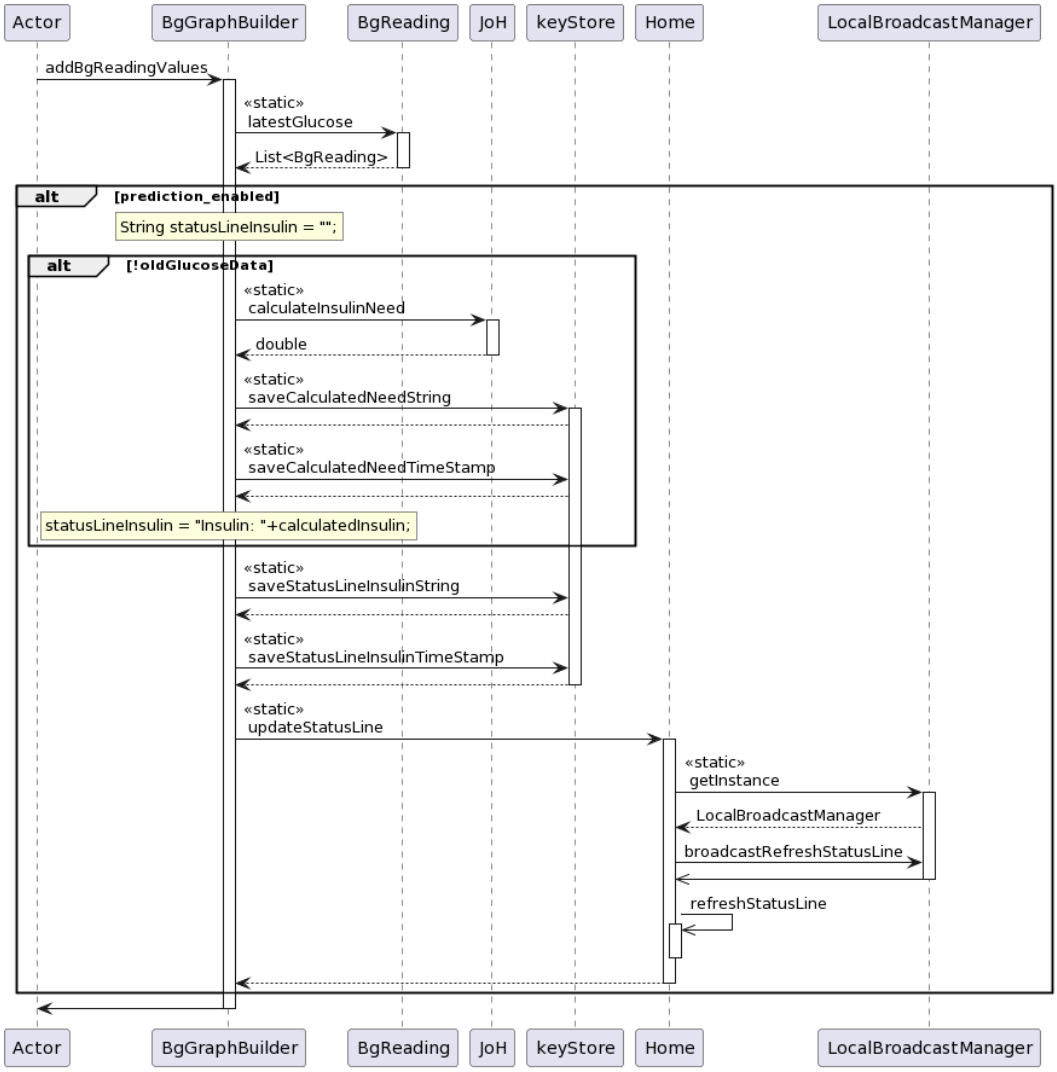
xDrip aplikazioan defektuz inplementatutako *intsulina* gomendioa sistema aztertuko da (formula matematikoan oinarrituta dagoena). Azterketa honek, ondoren *RL* bidezko gomendio sistema diseinatzeko balioko du.

Hasierako analisian ikusi den bezala (ikus 6.1.1. atala) "addBgReadingValues" metodoa hasiera puntu bezala hartuko da, *xDrip*-en formula matematikoetan oinarritutako *intsulina* gomendio sistema metodo honen bidez gauzatzen delako (ikus 6.9. irudia).

```
public class BgGraphBuilder {
    ...
    private synchronized void addBgReadingValues(final boolean simple) {
        ...
        if (prediction_enabled && simulation_enabled) {
            ...
            if (!BgReading.isDataStale()) {
                if (...) {
                    if (...) {}
                    else if (...) {
                    keyStore.putS("bwp_last_insulin", JoH.qs(evaluation[1],
                        1) + ((low_occurs_at > 0) ? ("!") : ""));
                    keyStore.putL("bwp_last_insulin_timestamp", JoH.tsl());
                    bwp_update = "\u224F" + " Insulin: " +
                        JoH.qs(evaluation[1], 1) + ((low_occurs_at > 0) ? ("
                        " + "\u26A0") : "");
                    }
                }

            }
            keyStore.putS("last_bwp", bwp_update);
            keyStore.putL("last_bwp_timestamp", JoH.tsl());
```

[5]Android Garatzaileentzako informazioa "PreferenceActivity" arabera: https://developer.android.com/reference/android/preference/PreferenceActivity



Irudia 6.9: "addBgReadingValues"-en sekuentzia diagrama.

```
            Home.updateStatusLine("bwp", bwp_update);
        }
        ...
    }
    ...
}
```

Metodo honek pertsonaren *gluzemia* balioak lortuko ditu (aplikazioaren datu basetik) eta grafiko bat sortuko du balio horiek erakutsiz. Prozesuan, "predictive simulations" funtzioa aktibatuta badago, beharrezkoa den *intsulina* edo karbohidrato kantitatea kalkulatu eta honekin interfazean erakutsiko den esaldia sortzen du. Ondoren 'Home.updateStatusLine("bwp", bwp_update);' sententziaren bidez eguneratzen da.

Eguneraketa egiteko, "Home" klasean "Intent" bat sortzen da, Android-ek erabiltzen duen "broadcast" sistema. "Home" klase berdinean, "OnCreate" metodoan, "BroadcastReceiver" bat dago, non "broadcast"-a jasoko den. Broadcast-ak "key" bat eta string bat du. String-aren balioa klasearen aldagai batean gordeko da eta "refreshStatusLine();" metodoa deituko da. Hau, bukatzeko, "status" barran dagoen esaldia sortuko eta interfazean erakutsiko da.

```
public class Home extends ActivityWithMenu implements ... {
    ...
    public static void updateStatusLine(String key, String value) {
        final Intent homeStatus = new Intent(Intents.HOME_STATUS_ACTION);
        homeStatus.putExtra(key, value);
        LocalBroadcastManager.getInstance(xdrip.getAppContext()).sendBroadcast(homeStatus);
        Log.d(TAG, "Home Status update: " + key + " / " + value);
    }
    ...
    protected void onCreate(Bundle savedInstanceState) {
        ...
        statusReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {

                final String bwp = intent.getStringExtra("bwp");
                if (bwp != null) {
                    statusBWP = bwp;
                    refreshStatusLine();
                } else {...}
            }
        };
        ...
    }
    ...
```

```
    private void refreshStatusLine() {
        try {
            String status = ((statusIOB.length() > 0) ? ("IoB: " +
                statusIOB) : "")
                    + ((statusBWP.length() > 0) ? (" " + statusBWP) : "");

            status = status + ((status.length()>0) ? " \u224F " +
                rl_insulin_need : rl_insulin_need); // RL insulin need. It
                will be empty if RL is not enabled
            ...
        } catch (NullPointerException e) {
            ...
        }
    }
}
```

6.2 Diseinua

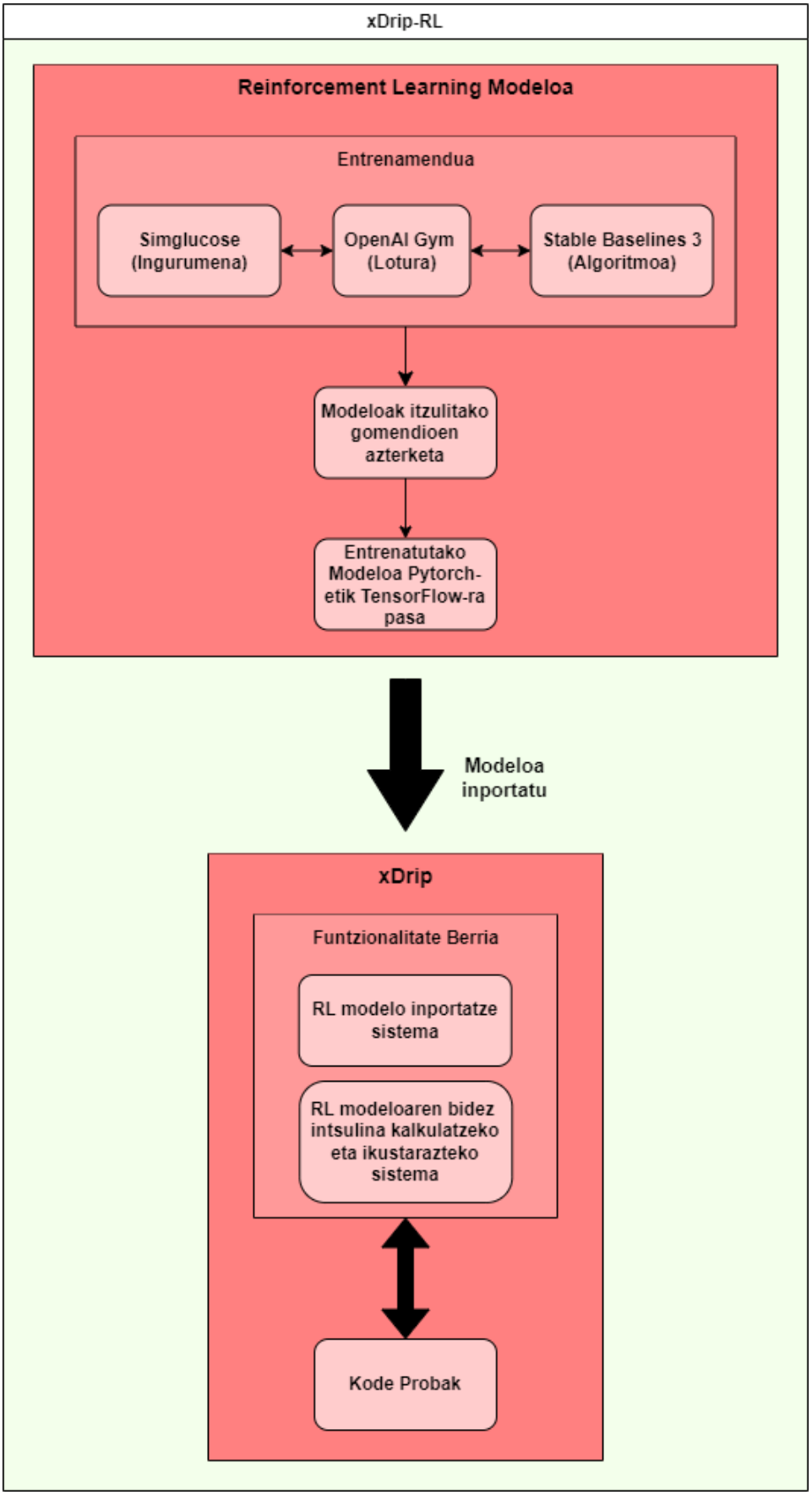
Atal honetan funtzionalitatearen garapen diseinua, kode berriaren diseinua eta sekuentzia diagramak azalduko dira.

6.2.1 Intsulina gomendio funtzionalitatearen diseinua

Atal honen bidez funtzionalitatearen diseinu eta egitura orokorra azaltzen da. 6.10. irudian diseinuaren diagrama ikusi daiteke. Funtzionalitateak bi garapen desberdindu ditu:

- **Funtzionalitatearen inplementazioa**: *xDrip* aplikazioaren kodean egindako aldaketa eta gehikuntzak dira. Betekizunetan adierazi den bezala, *RL* modelo bat inportatzeko eta erabiltzeko kodea garatu da. Ondoren kodearen probak egin dira. Proba hauek eskuz eta JUnit[6] bidez egin dira.
- **Modelo garapena**: *RL* modeloa sortzea da. Modeloa *xDrip* aplikazioan erabiliko da *intsulina* gomendioak kalkulatzeko. 7.3. atalean adierazten den bezala, diabetiko baten *intsulina* beharrak simulatzen dituen simuladore bat erabiliko da. *Simglucose* liburutegiko RL modeloak simuladorearen gainean entrenatu dira *intsulina* gomendioak emango dituen agentea lortzeko. Ikasketa bukatzean modeloak itzultzen diren gomendioak aztertu dira (egindako gomendioak onak ote diren ikusteko). Ikasketa eta probak (Pytorch framework-ean garatuak) bukatu eta gero modeloa *Tensor Flow* framework-era esportatu behar da Android-en erabiltzeko.

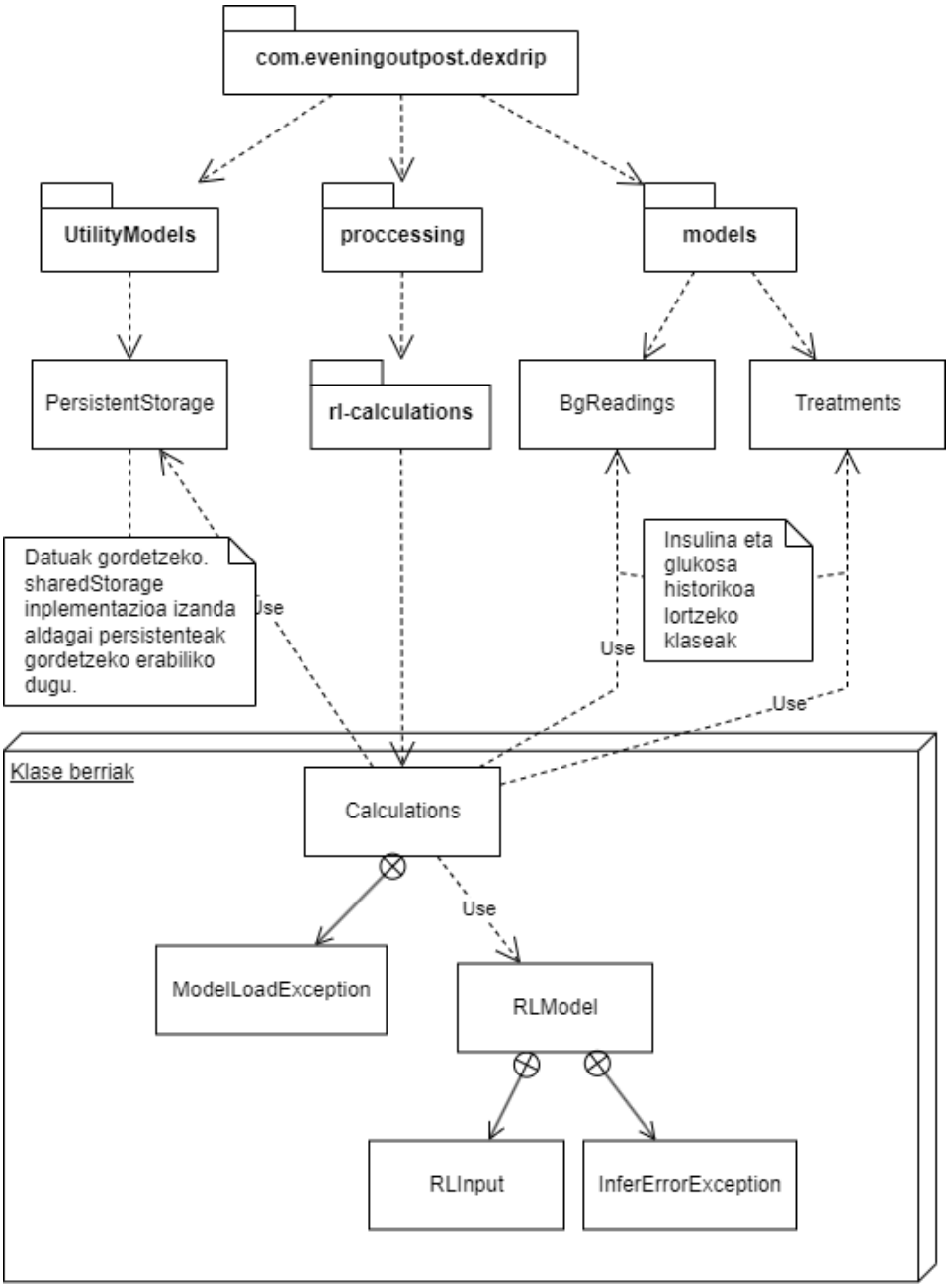
[6]Java kodearen probentzako sortutako liburutegia. Java kodea exekutatu eta aurretik idatzitako baieztapenen bidez emaitzak konprobatuko ditu.



Irudia 6.10: Funtzionalitate berriaren eta modeloaren diseinu eta pausoen diagrama.

6.2.2 Kodearen diseinua

Funtzionalitate berria inplementatzeko diseinua azalduko da (ikus 6.11. irudia). Bi klase berri sortu dira:



Irudia 6.11: Garatutako klase berrien diseinua.

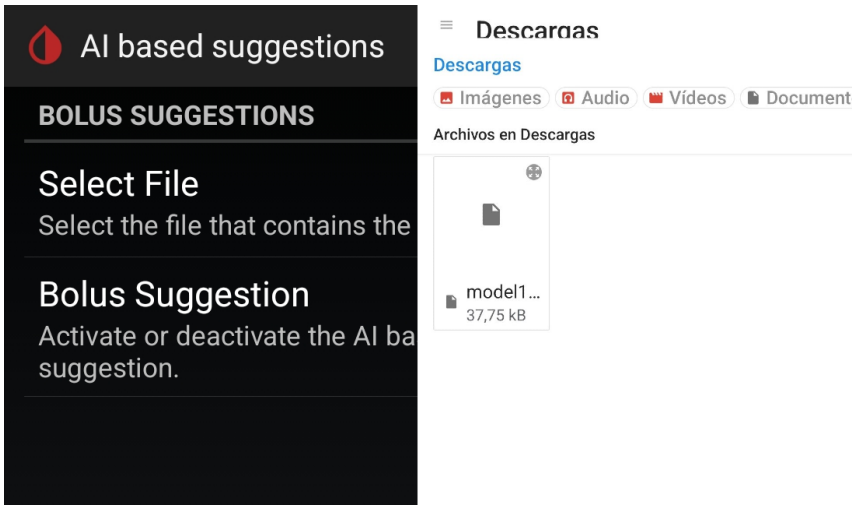
- **Calculations**: bere helburua aplikazioan dauden datuak lortzea, "RLModel" erabiliz emaitzak prozesatzea eta *xDrip*-en egin behar diren atazez arduratzea da (konfigurazioan gordetako datu baten eguneraketa adibidez). Ataza garrantzitsuenetariko bat *RL* modelo fitxategia inportatzea da. "RLModel" jaurtitzen dituen salbuespenak maneiatuko ditu.
- **RLModel**: *RL* modeloaren bidez kalkuluak egiteko helburua du. Klase bat izango du barnean, "RLInput". "RLInput" klaseak aplikazioan jaso diren datuak hartu eta prozesatuko ditu ondoren modeloak datu hauek erabili ahal izateko. Modeloa erabiltzean sortutako salbuespenak harrapatu eta salbuespen propio baten bidez kudeatuko ditu errore ezberdinak hobeto desberdintzeko.

6.2.3 Funtzionalitatearen sekuentzia diagramak

Intsulina gomendio sistemaren bidez *RL modeloa* inportatu eta modeloaren kalkuluak ikustaraziko dira. Funtzio hauen sekuentzi diagramak erakutsiko dira.

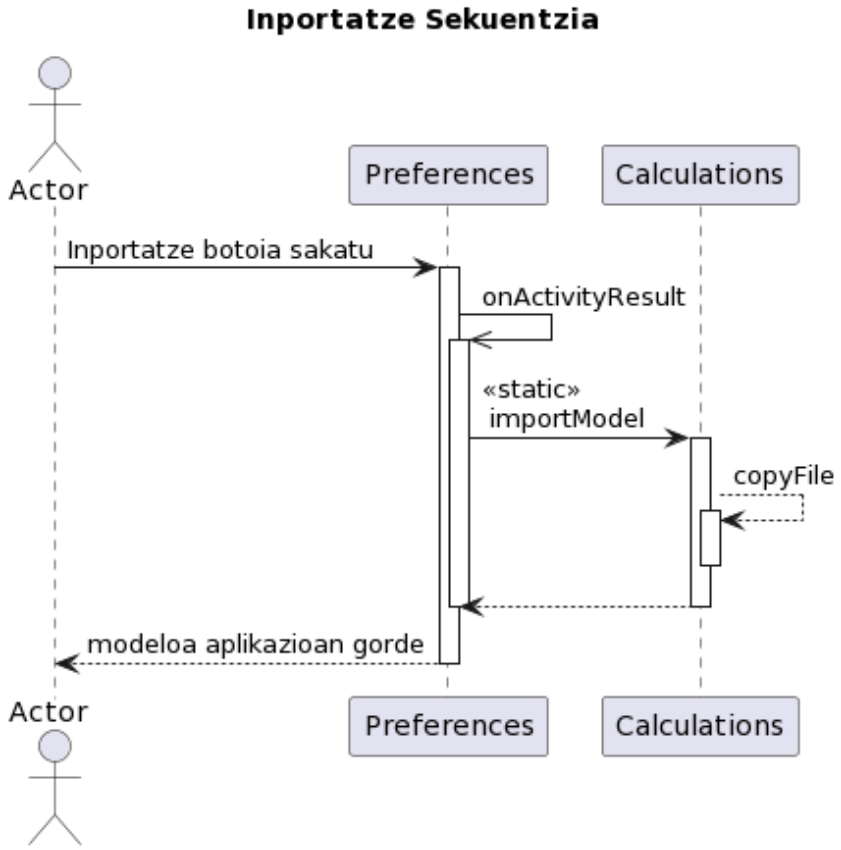
RL modeloa inportatu

Hasteko, *RL* modeloaren fitxategia mugikorraren memorian egon behar da ("Downloads" karpetan adibidez). Modeloa inportatzen hasteko konfigurazio interfazean inplementatuko den botoia sakatuko da (ikus 6.12. irudia). Ondoren Android-ek ematen duen fitxategi bilaketa sistemaren bidez modeloaren kokalekua (mugikorraren memorian) lortuko da.



Irudia 6.12: Modeloa inportatzeko sakatu behar den botoia eta fitxategi bilaketa interfazea.

Kokalekua "Calculations" klasearen "importModel"-ri pasatuko zaio. Metodoak mugikorren memorian dagoen fitxategia *xDrip* aplikazio memorian gordeko du. Ikusi 6.13. irudiko sekuentzia diagrama.



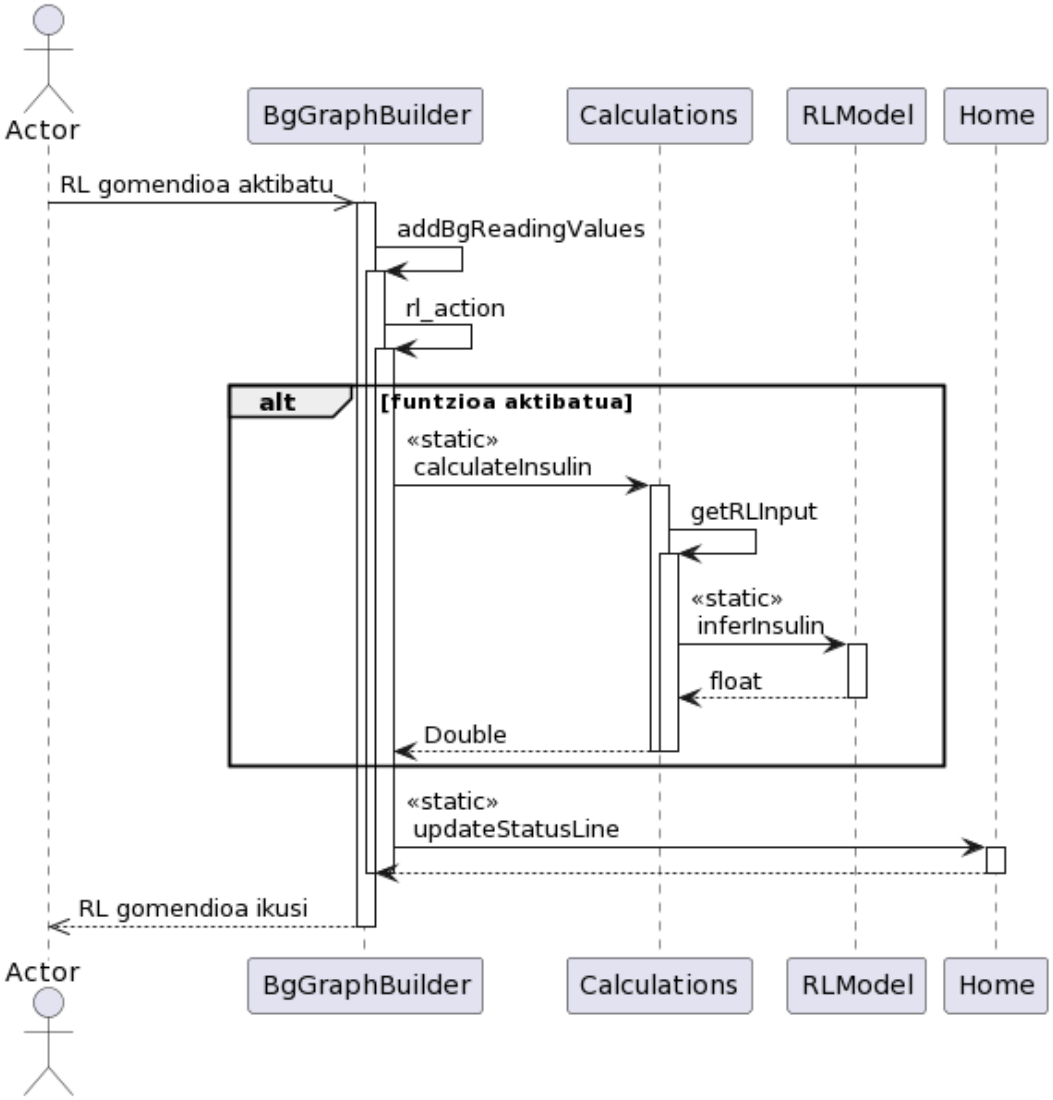
Irudia 6.13: Modeloa inportatzearen sekuentzia diagrama.

***RL* modeloaren emaitza ikustaraztea**

Aplikazioaren interfaze nagusia hasieratu bada, "BgGraphBuilder" klasea sortuko da eta "addBgReadingValues" bidez *gluzemia* balio berri bat lortzen den bakoitzean *intsulina* beharren kalkuluak egiten dira. Metodo honen barruan "rl_action" metodoa deituko da.

"rl_action" bidez (konfigurazioan *RL* bidezko gomendioa aktibatu bada) "Calculations"-eko "calculateInsulin" metodoa deituko da. Kalkuluak egiteko behar diren datuak lortu eta modeloa erabiliz kalkuluak egingo dituen "inferInsulin" metodoa deituko da. Bukatzeko, emaitza jaso eta gero interfazea eguneratuko da "updateStatusLi-

ne" bidez. Ikus 6.14. irudiko sekuentzia diagrama.



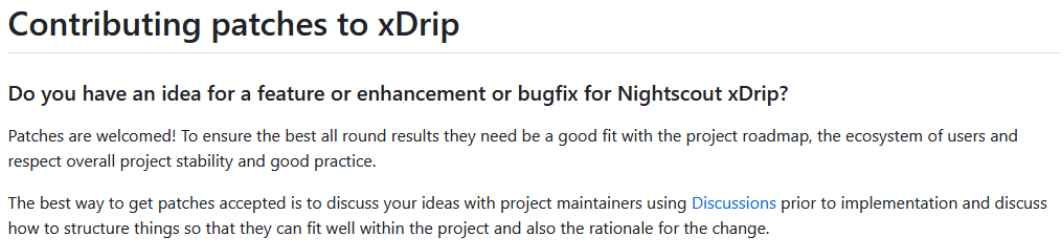
Irudia 6.14: *RL* gomendioa bistaratzen sekuentzia diagrama.

7 Proiektuaren garapena

Atal honetan *GrAL*-aren garapenean emandako pausuak azalduko dira. Hasteko garatzaileekin izandako komunikazioak azalduko dira. Ondoren funtzionalitatearen azalpenak bi zatitan banatuko dira, funtzionalitatearen inplementazioa aplikazioan eta *RL modeloa*ren garapena.

7.1 xDrip: Komunikazioak garatzaileekin

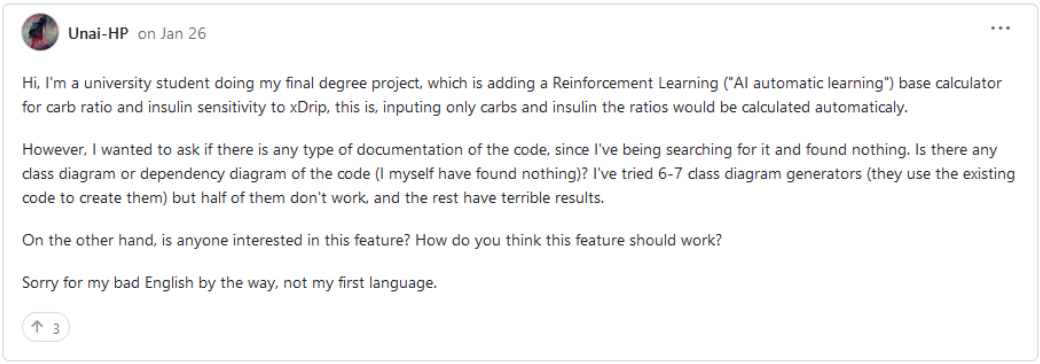
xDrip-en ekarpenak egiteko gidan adierazten den bezala (ikus 7.1. irudia), hedapena garatzaileekin sortutako *discussion*[1] atalean eztabaidatu da. Hasteko, hasierako ideia azaldu eta *xDrip*-en dokumentaziorik dagoen galdetu da (ikus 7.2. irudia). Garatzaileek hedapena onartu eta hasierako inplementazioa egiteko iradokizunak eman dituzte (ikus 7.3. irudia). Iradokizunetan aipatzen diren klaseak azkeneko inplementazioan erabili dira. Garatzaileek emandako informazio honek kodean bilaketa lana murriztu du eta denbora asko aurreztu du.



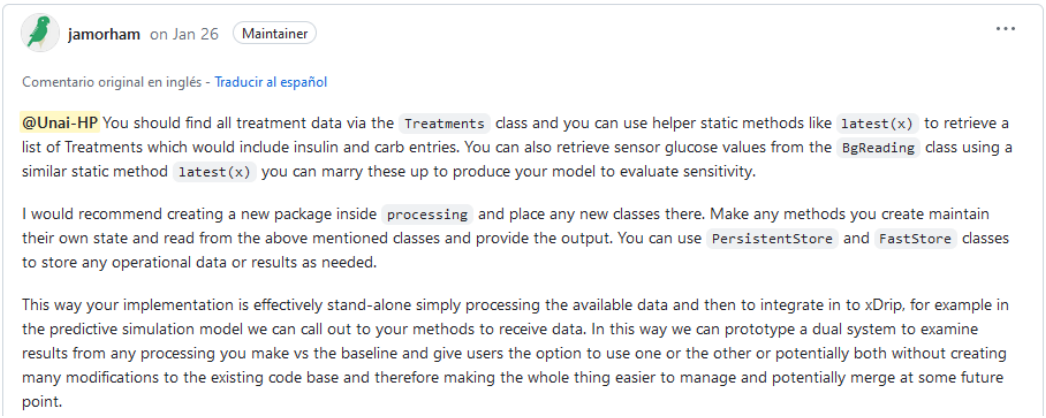
Irudia 7.1: *xDrip*-en ekarpen gidaren zati bat.

Beste aldetik, funtzionalitate berriaren interfazea garatzeko orduan arazo larri bat sortu da. Kodean interfazea aldatu eta konpilatu eta gero, aldaketak ez ziren aplikatzen. Denbora luze bat eman eta gero, garatzaileei arazoari buruz galdetzea erabaki eta *discussion* atal berdinean hari berri bat sortu da. Garatzaile batek arazo berdina izan duenez proiektuaren arazo bat dela suposatu daiteke (ikus 7.4. irudia). Interfazearen eguneraketa aplikatzeko gomendatzen dituen pausuak eman

[1] Eztabaida: https://github.com/NightscoutFoundation/xDrip/discussions/2651



Irudia 7.2: Garatzaileei bidalitako lehenengo mezua.



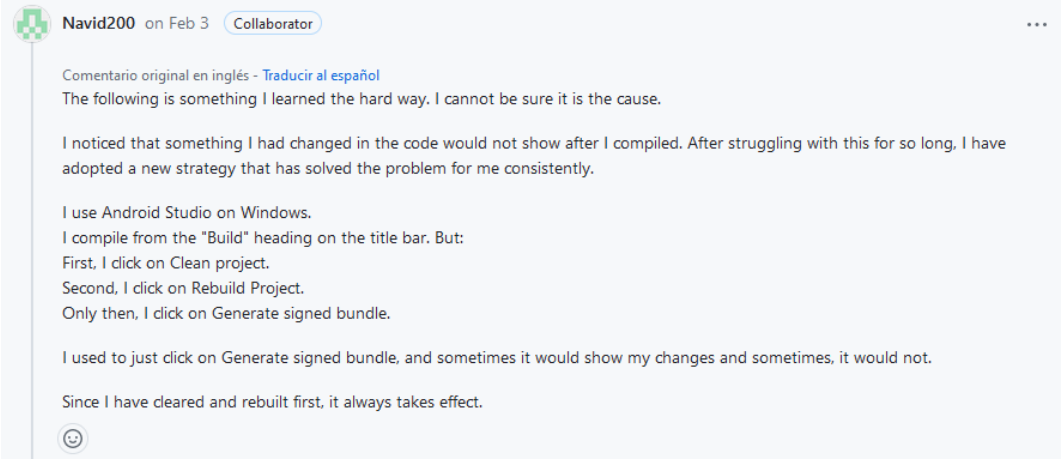
Irudia 7.3: Garatzaile nagusiaren lehengo erantzuna.

eta gero interfazea eguneratzea lortu da.

Esan beharra dago, hedapen berria guztiz garatzea lortu ez denez (*RL* modeloaren emaitzak ez dira behar bezain onak) *pull request* ez egitea erabaki dela. Horren ondorioz ez da garatzaileen feedback-a lortu.

7.2 Funtzionalitatearen inplementazioa xDrip aplikazioan

Funtzionalitateak *RL* modelo bat eta pertsonaren *gluzemia* erabiliko du pertsonak behar duen *intsulina* gomendioa kalkulatzeko eta erakusteko. Aldaketen azalpena bi zatitan banatu da, aplikazioari gehitutako klase berriak eta aplikazioan jada zeuden klase eta metodoen aldaketak. Atal bakoitzean aldatu edo gehitutako kodea azalduko da.



Irudia 7.4: *xDrip* garatzaile baten erantzuna interfazea eguneratze arazoei buruz.

7.2.1 Gehitutako klaseak

Bi klase berri sortu dira:

1. **Calculations**: Klase honen helburua *RL* modeloaren eta aplikazioaren arteko lotura egitea da. Aplikazioaren beste klaseek modeloarekin lan egiteko metodoak ditu, adibidez beharrezkoa den *intsulina* inferentzia egin nahi bada, klase honi deitu beharko zaio.

   Klase honek "RLModel" klase bat instantziatuko du eta gomendioak kalkulatzeko erabiliko da. Beste funtzio batzuk ditu, hala nola *modeloa*k behar dituen datuak lortzea eta *RL* modeloaren fitxategia inportatzea.

2. **RLModel**: modeloa maneiatzea eta erabiltzea da klasearen helburua. Modeloaren sarrerak definitzen ditu, eta sarrera hauek jasota *RL modeloa*ren bidez kalkuluak egiten ditu.

Calculations

Klase hau egiteko orduan, jakinda aplikazio osoan instantzia bakar bat beharrezkoa dela, "Singleton" patroia erabili da. Patroi honen bidez klasea deitzen den lehen aldian klasea instantzitu eta gordeko da. Hurrengo deiek gordetako instantzia hori erabiliko dute eta beraz, instantzia bakarra egongo da aplikazio osoan.

Kodearen egitura orokorra:

```
public class Calculations {
    private static Calculations instance;
    private RLModel model;
    // Beste atributu pribatuak
```

```
    public static Calculations getInstance() throws ... {...}
    private Calculations() throws ... {...}

    public float calculateInsulin() throws ... {...}
    public Double calculateBasal() throws ... {...}

    // Modeloak behar dituen datuak lortu aplikazioaren datu basetik
    private RLModel.RLInput getRLInput() {...}
    // Beste laguntza metodo pribatuak

    public void importModel(Uri uri) throws ... {...}
    // Inportatze prozesuan erabiltzen
    // diren metodo pribatuak
}
```

Klase honetan dauden "calculate" motako metodoetatik, momentuz, "calculateInsulin" metodoa bakarrik inplementatu da. Beste metodoa etorkizunean egiteko definitu da.

Kodearen adibide bat ikusteko, "getRLInput" erakutsiko da:

```
private RLModel.RLInput getRLInput(int size) {
    // Lortu gluzemia irakurketak
    List<BgReading> bgreadings = BgReading.latest(size);

    ArrayList<RLModel.RLInput.DataPoint> dataPoints = new ArrayList<>();

    // Irakurketa bakoitzeko data puntu bat sortu eta irakurketa
    // eta tratamenduak gorde.
    for (BgReading bgreading : bgreadings) {
        RLModel.RLInput.DataPoint dataPoint = new
            RLModel.RLInput.DataPoint();
        dataPoint.bgreading = (float) bgreading.calculated_value;
        dataPoint.timestamp = bgreading.timestamp;

        Treatments treatment = Treatments.byTimestamp(bgreading.timestamp);
        if (treatment != null) {
            dataPoint.carbs = (float) treatment.carbs;
            dataPoint.insulin = (float) treatment.insulin;
        }
        dataPoints.add(dataPoint);
    }

    // RLInput klaseari eman eta klasea itzuli
    return new RLModel.RLInput(dataPoints);
}
```

Kode honen bidez *gluzemia* mailaren balio multzo bat ("size" parametroaren arabera) lortuko da. Ondoren, *gluzemia* maila balio bakoitzeko une berdinean emandako tratamenduak (*intsulina* edo karbohidratoak) lortu eta guztia "DataPoint" batean gordeko da. "RLInput" klaseak "DataPoint" zerrenda bat jasoko du *RL modeloa*k erabiltzeko.

RLModel

Entrenatutako modelotik emaitzak lortzeko erabiltzen da. Gainera, klasean bertan, metodoentzako sarrera bezala erabiliko den klasea du definituta.

```
public class RLModel {
    public static class InferErrorException extends Exception {}
    private final Interpreter interpreter;

    public RLModel(Interpreter interpreter) {...}

    public float inferInsulin(RLInput input) throws ... {...}

    public static class RLInput {...}
}
```

"inferInsulin" metodoa bolo sistema adimentsuan erabiltzeko metodoa da. Metodoak "RLInput" klasea jasoko du. Klase honek modeloak beharko dituen datuak gordeko ditu, datuen "wrapper" bat bezala lan eginez. Datu hauek "RLModel" klasean bakarrik erabiliko direnez, komenigarria da "RLInput" barnean egotea, horrela kodearen hurbiltasuna handiagoa delako eta kodearen irakurketa errazagoa delako. Gainera datu prozesaketa aldatu nahi bada, klase hau aldatu beharko da bakarrik.

```
public static class RLInput {
    ArrayList<DataPoint> dataPoints;

    static class DataPoint {
        public float bgreading;
        public float insulin;
        public float carbs;
        public long timestamp;
    }

    public RLInput(List<DataPoint> dataPoints) {...}

    public float getLatestBG() {...}

    // Beste ezaugarrietan erabiliko diren metodoak
}
```

Klase honen bidez denbora puntu ("timestamp") batean jaso den *gluzemia* maila (bgreading) eta erabili diren *intsulina* eta karbohidrato (carbs) kantitateak gordeko dira.

7.2.2 Aplikazioaren aldaketak

Bi zatitan banatu dira aldaketak, funtzionalitateen aktibazioa (eta honen bidez emaitza erakustea) eta modeloaren inportazioa (*RL* modeloa aukeratu eta aplikazioan kargatu). Laburki kodearen aldaketak azalduko dira.

Funtzionalitatea aktibatzea eta ikusaraztea

Erabiltzaileak funtzionalitatea interfazearen bidez aktibatuko du. 6.1.1 atalean adierazten den bezala, aplikazioaren interfazearen xml-a ("xdrip_plus_prefs") aldatu behar da. Hasteko betekizunetan (ikus 5.4. atala) adierazitako 'AI based seuggestions' konfigurazio eremua/interfazea sortu da. Bere barnean, "PrefrenceCategory" erabiliz, *intsulina* gomendio sistema aktibatzeko etengailua (edo "switch") eta *RL* fitxategia inportatzeko botoia gehitu dira.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="..." xmlns:app="...">
    <PreferenceCategory
        android:key="xdrip_plus_experimental"
        android:title="@string/title_xdrip_plus_experimental">
        ...
        <PreferenceScreen
            android:key="ai_screen"
            ...
            android:title="AI based suggestions">
            <PreferenceCategory
                ...
                android:title="Bolus suggestions">
                <Preference
                    ...
                    android:title="@string/title_rl_file_picker" >
                </Preference>
                <SwitchPreference
                    android:defaultValue="false"
                    android:key="rl_simulations_enabled"
                    ...
            </PreferenceCategory>
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>
```

"SwitchPreference" XML adabegiaren bidez konfigurazioan etengailu bat agertuko da eta honetan klik egitean bere "key" balioa aldatuko da. Balio honen aldaketaren arabera interfaze nagusian emaitza erakutsiko da.

Hau egiteko "BgGraphBuilder" klaseko "addBgReadingValues" metodoa aldatu da. Metodo honi *gluzemia* balio bat lortzen denean deitzen zaio. Metodo honetan formula matematikoaren bidez kalkulatzen den *intsulina* beharra eguneratzen da.

```
private synchronized void addBgReadingValues(final boolean simple) {
    ...
    try {
        ...
        if (!simple){
            ...
            // RL bidez intsulina beharra kalkulatu gluzemia balio berria
            // lortzen denean.
            try { rl_action(); }
            catch (Exception e) {
                Log.e(TAG, "Exception in RL prediction: " + e.toString());
            }
        }
    }
}
```

"rl_action" metodoak (funtzioa aktibatu bada, hau da, "rl_simulations_enabled=true" bada), "Calculations" instantzia deitu eta beharrezkoa den *intsulina* kalkulatuko du. Emaitza gorde eta aplikazioaren interfaze nagusian ikustaraziko da balioa.

```
private void rl_action() {
    String insulinDisplayed = "";
    if (prediction_enabled && rl_simulation_enabled) {
        try {
            float calculated_insulin =
                Calculations.getInstance().calculateInsulin();
            insulinDisplayed = "insulin(RL): " + calculated_insulin;
        }
        catch (Calculations.ModelLoadException e) {
            ...
        }
        catch (RLModel.InferErrorException e) {
            ...
        }
    }
    keyStore.putS("rl_insulin_need", insulinDisplayed);
    Home.updateStatusLine("insRL", insulinDisplayed);
}
```

Modeloa inportatu

Intsulina sistema adimentsuak emaitzak sortzeko *RL* modelo bat izatea beharrezkoa da. Modelo hori aplikaziotik kanpo garatuko da. Aplikazioak ez du defektuzko modelorik izango beraz modeloa inportatzen duen kodea (modeloa mugikorraren memorian bilatu eta aplikazio memoriara kopiatu) inplementatu behar da.

Modeloa inportatzeko aplikazioaren konfigurazio aldaketak maneiatzen dituen "Preferences" klasea aldatu da. Klasea sortzen den bakoitzean "setOpenTFliteFile()" metodoa deitu eta inportatze botoiaren entzule bat sortuko du. Entzulearen bidez, botoia sakatzen den bakoitzean interfaze grafiko bat erabiliz erabiltzaileak fitxategi (URI) bat bidaliko dio aplikazioari.

```
private void setOpenTFliteFile() {
    // Konfigurazio elementua lortu
    Preference prefTfliteFile = findPreference("rl_file_picker");

    // Klik egiten den bakoitzean ACTION_OPEN_DOCUMENT hasi
    prefTfliteFile.setOnPreferenceClickListener(preference -> {
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.setType("*/*");

        startActivityForResult(intent, Constants.RL_MODEL_FILE_INTENT_ID);
        return true;
    });
}
```

"Preferences"-en edozein "Intent" (fitxategia irekitzeko prozesua adibidez) bukatzen den bakoitzean "onActivityResult(...)" metodoak emaitzak jasoko ditu. Hainbat "Intent" mota jaso ahal dira, beraz "Intent" bakoitzak String identifikadore bat izango du. Inplementatu den "Intent"-aren identifikazioa "Constants" klasean definitu da. Identifikazioa zuzena bada eta prozesuak errorerik izan ez badu, fitxategi baten URI-a jaso, gorde eta "Calculations" klasearen bidez erabiliko da (fitxategia aplikazioaren memorian kopiatzeko eta modeloa kargatzeko).

```
public void onActivityResult(int requestCode, int resultCode, Intent data)
    {
    ...
    if (requestCode == Constants.RL_MODEL_FILE_INTENT_ID && resultCode ==
        Activity.RESULT_OK) {
        if (data != null) {
            String path = data.getData().getPath();

            PersistentStore.setString(Constants.RL_MODEL_FILE_PATH, path);
            try { Calculations.getInstance().importModel(uri); }
            catch (IOException IOE) {...}
```

```
        }
    }
}
```

7.3 Modeloaren entrenamendua

Atal honetan *intsulina* kopurua gomendatuko duen *RL modeloa*ren entrenamendua azalduko da. Bi zatitan banatu da entrenamendua, hasierako inplementazioa (modeloa garatzeko tresnak eskaintzen zituzten defektuzko algoritmoak erabili dira) eta ikerketetan oinarritutako algoritmoaren bidez entrenatutako modeloa. Lehenengo atalean erremintak azalduko dira gehien bat, eta bigarrenean erabilitako algoritmo azalpena emango da.

7.3.1 Hasierako inplementazioa

Modeloaren entrenamendua egiteko *Simglucose* (*SG*), *OpenAI Gym (Gym)* eta *Stable Baselines 3* (*SB3*) erabili dira. *SG* diabetesa duten gaixoen simulagailua da. Gaixo hauei *intsulina* gomendatzen dien *RL* modeloa sortzeko *SB3* liburutegiaren algoritmoak erabili dira.

SG eta *SB3* ez daude haien artean lan egiteko prestatuta. Hala ere biek *Gym API*-a dute inplementatuta. *Gym*-en bidez simulagailua eta algoritmoaren arteko lotura egin daiteke, irteerak eta sarrerak konektatuz.

Hala ere *Gym* erabiltzeak arazo bat sortzen du. *SG* simulagailuak *Gym*-en bertsio zaharkitua du inplementatua ("gym==0.9.4") eta *SB3*-ak beste aldetik bertsio berriagoa behar du (gymnasium[2] edo gutxienez "gym==0.21.0").

Bateraezintasun hau zuzentzea ez da erreza izan. Hasieran *Gym* bertsio desberdinak probatu dira *SG* eta *SB3* biekin aldi berean bateragarria den bertsioa bilatuz. Hainbat probatu dira eta guztiak bateraezinak direnez, *Gym* zaharrago bat erabiltzen duen *SB3* bat probatu da, arrakastarik gabe.

Beste hainbat eratan probatu eta gero *SG* proiektuaren *Gym* "wrapper"-a eguneratzea erabaki da. *Gym* "wrapper" egitura nola erabili behar den aztertzen denbora bat eman eta gero, ikusi da *SG*-en aldaketa sinple bat eginez bateraezintasuna konpondu daitekeela. Egin beharreko gauza bakarra "wrapper"-ean dauden metodoen izenen aldaketa da, hau da, metodoen aurrean dauden "_" karakterea kentzea.

...

[2] *OpenAI Gym*-ren ondorekoa den softwarea: https://gymnasium.farama.org/

```
# Hasieran zegoen metodo izena
def _step(self, action):
# Bateragarria den metodo izena
def step(self, action):
    ...

# Hasieran zegoen metodo izena
def _seed(self, seed=None):
# Bateragarria den metodo izena
def seed(self, seed=None):
    ...

# Hasieran zegoen metodo izena
def _create_env_from_random_state(self, custom_scenario=None):
# Bateragarria den metodo izena
def create_env_from_random_state(self, custom_scenario=None):
    ...

# Hasieran zegoen metodo izena
def _render(self, mode='human', close=False):
# Bateragarria den metodo izena
def render(self, mode='human', close=False):
    ...
...
```

Aldaketa hauek *SG*-ren *fork* batean egin dira eta hemendik aurrera hau erabili da. Bateragarritasuna zuzenduta modeloa entrenatzea posiblea izan da.

Modeloa entrenatzeko, *SB3*-n inplementuta dauden RL metodoak erabili dira. Ondorengo algoritmo hauek probatu dira: PPO[3], A2C[4], TD3[5], SAC[6]. Emaitza hoberenak eman dituen algoritmoa PPO[7] izan da.

RL modeloa garatzeko erremintak

Modeloa entrenatu ahal izateko eta honen emaitzak zuzenak diren aztertzeko hainbat Python script inplementatu dira:

1. **Training** script-a: *SB3* algoritmo eta *SG* simulagailuaren bidez *intsulina* beharra kalkulatzen duen modeloa entrenatzen duen script-a. Entrenamenduaren parametroak zehaztu eta ikasketa begizta hasieratzen du. Hurrengoa scriptaren laburpen bat da:

[3] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html
[4] https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html
[5] https://stable-baselines3.readthedocs.io/en/master/modules/td3.html
[6] https://stable-baselines3.readthedocs.io/en/master/modules/sac.html
[7] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

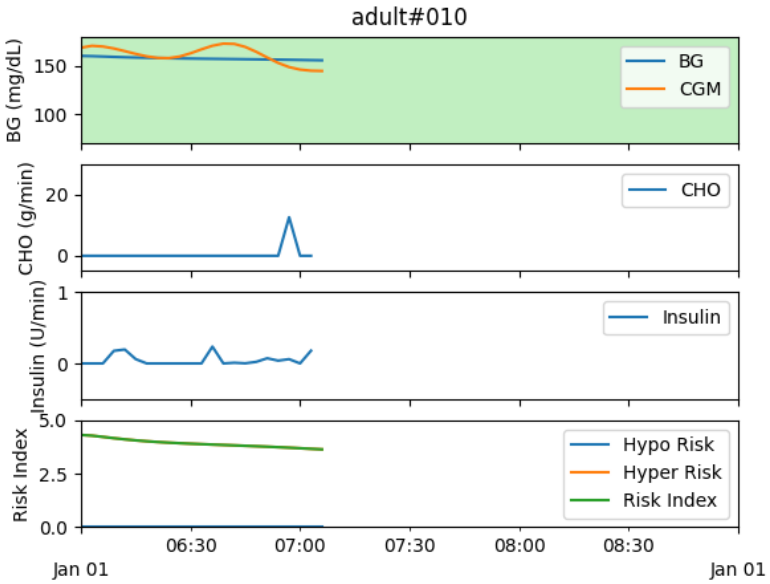
```
total_timesteps = 10_000_000
...
batch_size = 320
mini_batch_size = 20
PPO_epochs = 10
...

def train():
    hyperparams = {...}
    env = create_random_envs(num_envs)

    while(True):
        model.learn(...)
        model.save(...)

        model.set_env(...)


def create_random_envs(num_envs):
    # Create new vectorized environment for another set of patients
```



Irudia 7.5: *OpenAI Gym*-en "render" metodoaren emaitza. BG odoleko *gluzemia* erreala da. *CGM*-a *gluzemia* neurtzen duen sentsorea da (*CGM* sentsoreen baloiak errorea duten BG balioak dira). CHO pazienteak unean jan dituen karbohidratoak dira.

2. **Test** script-a: *RL* modeloaren probak egiteko script-a da. Modelo eguneratuena lortu eta bere gainean test-ak egiten dira. Test hauen emaitzak ikusi daitezke *Gym*-ren "render" metodoaren bidez (ikus 7.5. irudia). Hurrengoa script-aren laburpen bat da:

```
trainingFolder = '...'
def test():
    model_path = get_latest_model_file(trainingFolder)
    env = getRandomEnv()

    model = PPO.load(model_path, env=env, verbose=2)

    obs = env.reset()
    while True:
        action, i = model.predict(obs)
        obs, reward, done, info = env.step(action)
        env.render()
        if done:
            obs = getRandomEnv().reset()
```

3. **Konbertsio** script-a: modeloa *SB3* bidez garatu da, Pytorch-en oinarritua dagoena. Android-ek ordea *Tensor Flow*-rekin hobeto lan egiten du. Beraz modeloa Pytorch-etik *Tensor Flow*-ra bihurtzeko script hau sortu da. Hau egiteko *SB3*-ren gida ofiziala erabili da, baina gida ofiziala jarraituz hainbat arazo sortu direnez hauek zuzentzeko hainbat bilaketa egin dira. Hurrengoa script-aren laburpen bat da:

```
android_path = "android/"
model_prefix = "model"
model_path = android_path + model_prefix

onnx_save_file = android_path + model_prefix + ".onnx"
tflite_save_file = model_path + ".tflite"
tflite_quant_save_file = model_path + "_quant.tflite"

...

def model_to_onnx(model, onnx_path):
    # Convert Pytorch model to onnx (Open Neural Network Exchange) and
        save to file

def load_onnx(onnx_path):
    # Load onnx

def test_onnx(onnx_model, observation_size):
    # Test onnx

def onnx_to_tflite(onnxable_model, obs_space):
```

```
    # Convert onnx file to tflite file

def infer_latest_model(latest_model, bg):
    # Calculate/infer needed insulin with Pytorch file

def infer_onnx(onnx_model, bg):
    # Calculate/infer needed insulin with onnx file

def infer_tflite(tflite_file, input_data=150):
    # Calculate/infer needed insulin with tflite file

...

if __name__ == "__main__":
    latest_model = get_latest_model_file("...")

    # Use onnx
    model = load_model(model)
    model_to_onnx(model, onnx_save_file)
    onnx_model = load_onnx(onnx_save_file)
    test_onnx(onnx_model, observation_size)

    onnx_to_tflite(onnx_model, obs)

    infer_latest_model(latest_model, bg)
    infer_onnx(onnx_save_file, bg)
    infer_tflite(tflite_save_file, bg)
```

4. **Laguntzaile** script-a: aurreko script-etan kode errepikatua egon daiteke, adibidez script batzuk gaixo baten ingurunea lortu behar dute. Amankomunean duten kode errepikatu hau metodoetan multzokatu da eta laguntzaile script honetan gorde dira. Hurrengoa script-aren laburpen bat da:

```
def get_latest_model_file(folder_path):
    # Get path of file in OS

def load_model(model_path):
    # Load model to Pytorch

...

def make_enviroment(env_id, rank, reward_function=None, seed=0):
    # Creates patient for simulation

def get_random_enviroments(num_envs=10, reward_function =
    custom_reward, seed=0):
    patient_types = getPatients()
    patient_index_list = random.sample(...) # select a random subset
```

```
            of patients
        selected_patients = [patient_types[patient_index_list[i]] for i in
            range(0,num_envs)]

        # create an instance of each possible patient
        return [make_enviroment(...) for i in
            range(0,len(selected_patients))]

def getPatients():
    return ["adult#001", "adult#002", ...]

...

def getRandomEnv(reward_function = custom_reward):
    # Get a random patient
```

Modeloaren entrenamendua egin ahal izateko kalkulu zentroko konputagailu batean kontua sortu eta etxetik erabili da. Ordenagailuan lan egiteko docker[8] erabili da. Ordenagailuaren erabilera errazagoa izateko hainbat erreminta erabili dira:

1. **Docker-file**: docker erabiltzeko sistema irudiak beharrezkoak dira. Modeloa entrenatzeko kasuan ordenagailuaren GPU-a[9] erabili nahi da, entrenamendua azkarragoa izateko, eta beraz hau ahalbidetzen duen irudia bilatu da. Irudi honek ez ditu instalatuta entrenamendurako behar diren software guztiak (*SB3* adibidez), beraz docker-file baten bidez, irudi bat hartu eta honen gainean softwarea instalatu eta irudi berri bat sortu da, modeloa entrenatzeko erreminta guztiak dituena.

2. **Docker compose**: erreminta honen bidez docker edukiontziak sortu daitezke fitxategi batean ezaugarri guztiak adieraziz. Ezaugarri hauen artean aurretik sortutako irudia erabili da, baita GPU-a erabiltzeko baimena ere. Docker compose fitxategia sortu eta gero, edozein momentutan eta era erraz batean edukiontzia sortu eta aldaketak egin daitezke.

3. **Terminal script-ak**: modeloa entrenatzen denean garapenaren jarraipena egiteko script-ak dira. Hauetako batzuk docker edukiontzira konektatzeko script-ak eta modeloaren entrenamendu fitxategiak ordenagailu lokalera deskargatzeko script-ak dira.

7.3.2 Ikerketetan oinarritutako modeloa

*Modeloa*ren entrenamenduan algoritmoak defektuz uzteak (hiperparametroak, algoritmoan erabiltzen den sare neuronal arkitektura...) ez du errendimendu ona

[8] Docker software birtualizazio sistema bat da.

[9] "Graphics Processing Unit" grafikoak bizkor prozesatzeko edo koma higikorreko operazioak azkar kalkulatzeko erabiltzen den prozesagailu espezializatu bat da.

izan. Beraz ikerketetan arrakasta izan duen algoritmoren bat erabiltzen saiatzea erabaki da. *Intsulina* tratamendua kontrolatzen duten hainbat algoritmo aurkitu ahal dira [3, 2, 7].

Azkenean Lee et al. [7] ikerketan erabilitako algoritmoa aukeratu da . Laburbilduz, pankreak erabiltzen dituen portaera formulen hurbilpenak erabiltzen dira *RL* modelo bat sortzeko. Algoritmoaren elementu garrantzitsuenak hurrengoak dira:

- **β-zeluletan inspiratutako sari funtzioa**: pankreak dituen β-zelulak *intsulina* sintetizatzen eta jariatzen dute. Zelula hauen *intsulina* jariaketa formula baten bidez adierazi daiteke

$$SR(t) = SR_{basal} + SR_{above-basal}(t)$$

jakinda SR_{basal} basal *intsulina* jariaketa (ikus 4.1.2. atala) eta $SR_{above-basal}$ bolo *intsulina* jariaketa (ikus 4.1.2. atala) direla. Beste era batera esanda, SR_{basal} epe luzerako jarioa eta $SR_{above\text{-}basal}$ epe laburreko jaria izango dira. Ondorioz, *RL* algoritmoaren saria epe laburrekoan (R_{short}) eta luzekoan (R_{long}) banatu dira. Horrela definitu dira:

$$R_{long} = \begin{cases} -D_L(g) & if\ 70 \leq x \leq 180 \\ -3 \cdot D_L(g) & otherwise \end{cases} \tag{7.1}$$

$$R_{short} = \begin{cases} \begin{cases} -5 \cdot D_R(g,v) & if\ v < -0.6 \\ 0 & if\ v \geq 3 \\ -3 \cdot D_R(g,v) & otherwise \end{cases} & if\ g < 100 \\ \begin{cases} 0 & if\ v \geq 3 \\ -D_R(g,v) & otherwise \end{cases} & if\ 100 \leq g < 160 \\ \begin{cases} -5 \cdot D_R(g,v) & if\ v \geq 3 \\ -D_R(g,v) & otherwise \end{cases} & if\ 160 \leq g < 180 \\ \begin{cases} -5 \cdot D_R(g,v) & if\ v \geq 1.5 \\ -3 & otherwise \end{cases} & otherwise \end{cases} \tag{7.2}$$

non g uneko *gluzemia* den eta v uneko *gluzemia* eta aurrekoaren arteko desberdintasuna den. $D_L(g)$ uneko glukosa eta helburu glukosaren arteko desberdintasuna da. $D_R(g,v)$ (ikus 7.3. formula) nahi den glukosara heltzeko beharrezko gluzemia aldaketa da (100-eko *gluzemia* nahi bada eta 180 *gluzemia* izanda, v negatiboa izatea nahi dugu, adibidez -5 izatea).

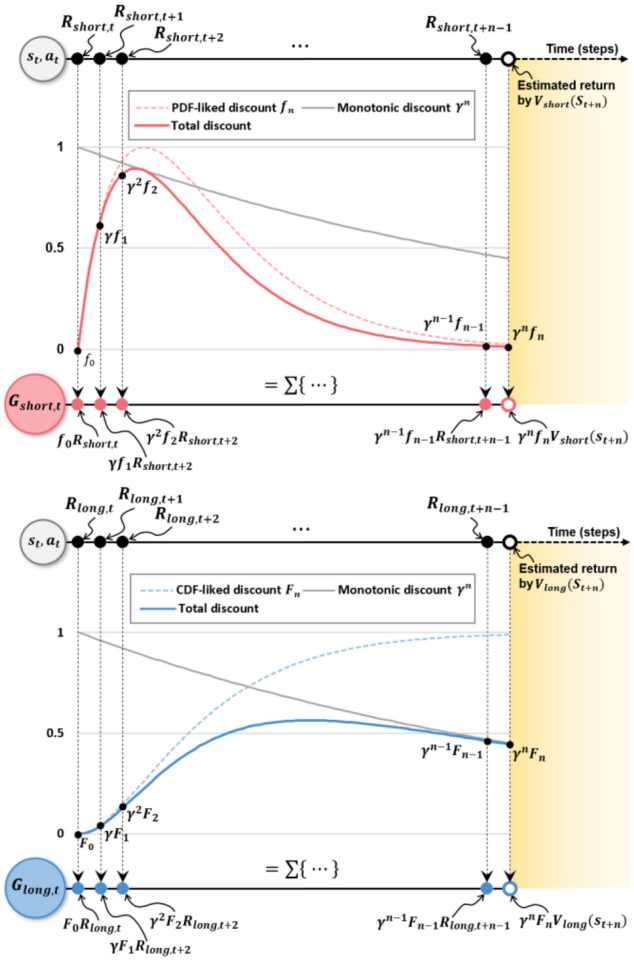
$$D_R(g,v) = |1/15 \times (g - G_{target}) - v| \tag{7.3}$$

Ikusi daitekeen bezala saria egoera onuragarrian, arriskutsuan eta oso arriskutsuan banatu dira, -1, -3 eta -5 pisuekin hurrenez hurren. Kasu desiratuenek (*gluzemia* baxua eta igotze handia, eta alderantziz) ez dituzte sari negatiborik.

- **Pertsona bakoitzaren farmakokinetika eta farmakodinamikan oinarritutako deskontu indizea**: *RL* algoritmoek sari metagarriaren arabera eguneratuko du bere politika. Beste era batera esanda, hainbat akzio egin eta gero, hauek lortu duten sari metagarria kalkulatu eta hau hobetzen saiatuko da. Iraganean lortutako sariek gehiegizko garrantzia ez izateko, γ balio bidez biderkatuko dira, non $\gamma \in [0, 1]$. Beraz formula hurrengoa izango litzateke:

$$G_t = R_t + \gamma R_{t+1} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n V(s_t + n) \tag{7.4}$$

Ikerketaren kasuan bi sari metagarri (ikus 7.6. irudia) definitu dira, epe labur edo luzeen arabera:



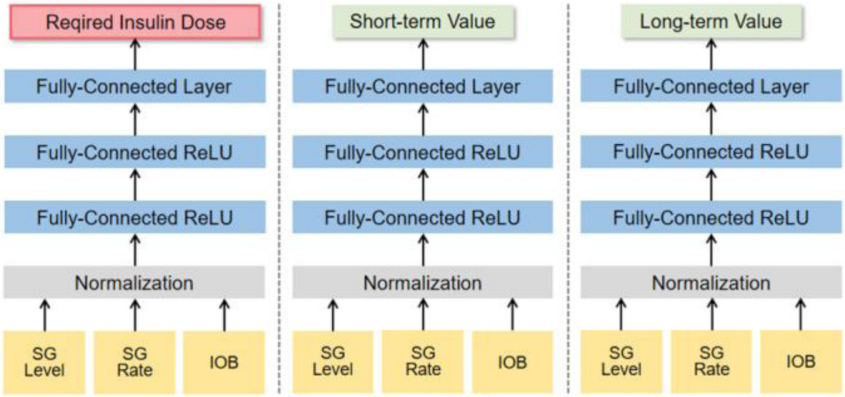
Irudia 7.6: Sari metatu bien irudikapen grafikoa.

$$G_{short,t} = f_0 R_{short,t} + \gamma f_1 R_{short,t+1} + \cdots + \gamma^{n-1} f_{n-1} \times R_{short,t+n-1} + \gamma^n f_n V_{short}(st+n), \tag{7.5}$$

$$G_{long,t} = F_0 R_{long,t} + \gamma F_1 R_{long,t+1} + \cdots + \gamma^{n-1} F_{n-1} \times R_{long,t+n-1} + \gamma^n F_n V_{long}(st+n), \tag{7.6}$$

$V_{short}(s_k)$ eta $V_{long}(s_k)$ epe laburrerako eta luzerako espero diren balioak dira S_k egoerarako. F_k terminoak *intsulina*ren metaketaren efektu famakodinamikoa simulatzen du eta f_k terminoak *intsulina* xurgatzearen propietate farmakokinetikoa simulatuko du. Bi termino hauek *Simglucose* simulazio eta test batzuen bidez kalkulatu dira.

- **Politika ikasketa**: politika ikasteko *Proximal Policy Optimization* (*PPO*) *RL* algoritmoa erabili da. Hiru sare erabili (ikus 7.7. irudia) dira: egoeraren arabera diabetikoak hartu beharko duen *intsulina* dosia kalkulatuko duen sare bat, eta V_{short} (G_{short}-ren itxarotako batezbestekoa) eta V_{long} (G_{long}-ren itxarotako batezbestekoa) balioak estimatuko dituzten beste bi sare. Bi balio hauek kalkulatutako *intsulina* dosiaren "advantage"-a (abantaila) kalkulatzeko erabiliko da, hau da, proposatutako dosia egokia den edo ez kalkulatzeko erabiliko da.



Irudia 7.7: Ikerketan erabilitako hiru sare neuronalak.

Sare neuronal bakoitzaren sarrera *gluzemia*, *gluzemia*n aldaketa (unekoaren eta aurrekoaren artean) eta "IoB" (jasotako eta aktibo dagoen *intsulina*) izan dira.

Entrenamenduari denbora bat eman zaio, baina ez dira emaitza onak lortu. Hipotesi bat algoritmoak denbora eta konputazio ahalmen handiagoak behar izatea da.

7.3.3 Modeloaren emaitzak

RL modeloaren entrenamendua bi eratan diseinatu da. Lehenengo era *Stable Baselines 3* liburutegiak eskaintzen dituen algoritmoak erabiliz izan da. Erabiltzeko errazak dira beraz lehenengo proba eta emaitzak algoritmo hauekin lortu dira. Bigarrena, emaitzak hobetzeko nahian, ikerketetan oinarritutako algoritmo batetaz baliatzea izan da.

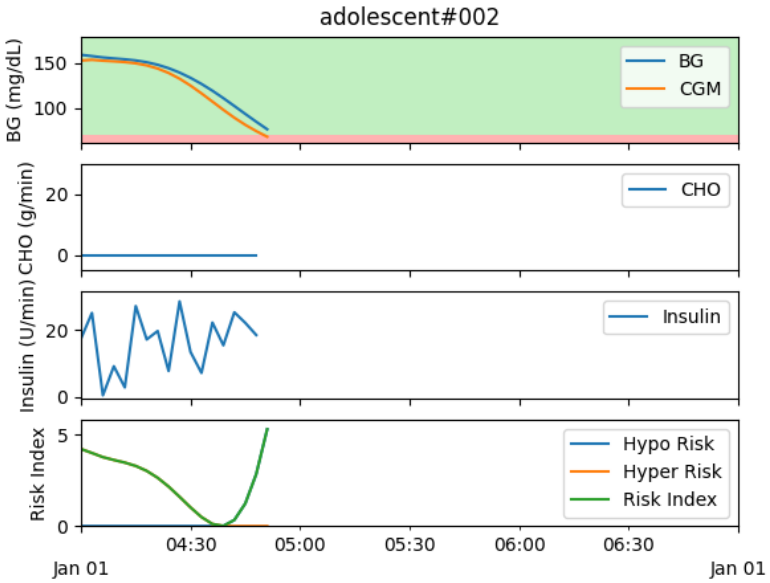
SB3 algoritmoen bidezko emaitzak

SB3-n inplementatuta dauden A2C[10], DQN[11], PPO[12] eta SAC[13] algoritmoak erabili dira. Entrenamendua algoritmo guztientzat 5 egunekoa izan da. Emaitza hoberenak dituen algoritmoa PPO[14] izan da.

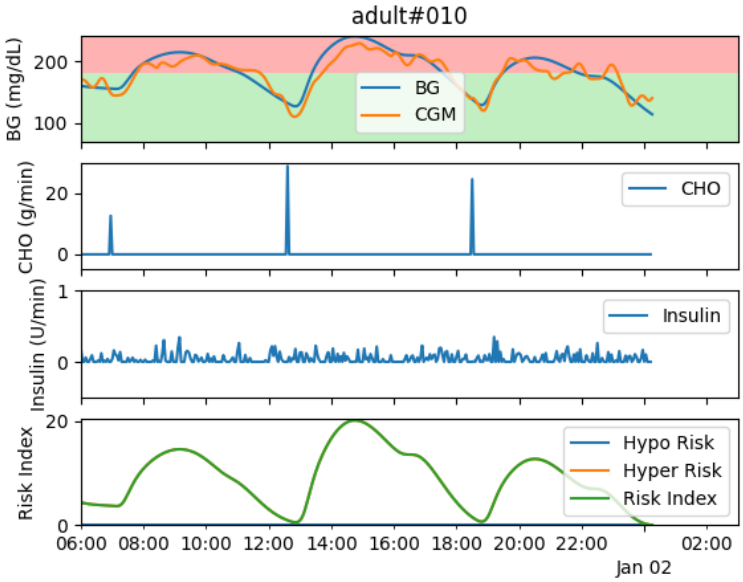
Emaitzak ikustarazteko *SG*-ren "render" metodoak sortzen duen simulazioaren irudia erabiliko da. Irudiak simulatutako pazientearen une bakoitzeko odoleko *gluzemia* (BG), *CGM*-ak neurtutako *gluzemia* (CGM), pazienteak jan dituen karbohidratoak (CHO) eta injektatutako *intsulina* erakusten ditu. Gainera egoeraren arriskua adieraziko duen neurketa (risk index) ikusi daiteke. Algoritmoaren helburua simulazioa ez bukatzea (gaixoa osasuntsu mantentzea) eta arrisku indizea gutxitzea da. Simulazioa *gluzemia* 65 mg/dl baino baxuagoa edo 350 baino altuagoa bada bukatuko da.

PPO ez ezik, beste algoritmo guztiek gehiegizko *intsulina* erabiltzen dute. Adibidez, 7.8. irudian ikusi daitekeen bezala A2C algoritmoak 20 unitate *intsulina* baino gehiagoko injekzioak egiten ditu. Hauek oso kantitate altuak dira, normalean pertsona batek orduro (barau egitean) ez du 1.5 unitate baino gehiago behar izaten. PPO algoritmoak *intsulina* hobeto maneiatzen du eta ondorioz simulazioa denbora gehiago mantentzen du. 7.9. irudian ikusi daitekeen bezala pazientea egun oso batean mantentzeko gaitasuna du. Hala ere pazientearen arabera emaitza desberdinak lortzen dira, eta adibidez nerabeak diren pazienteetan emaitzak ez dira onak (ikus 7.10. irudia).

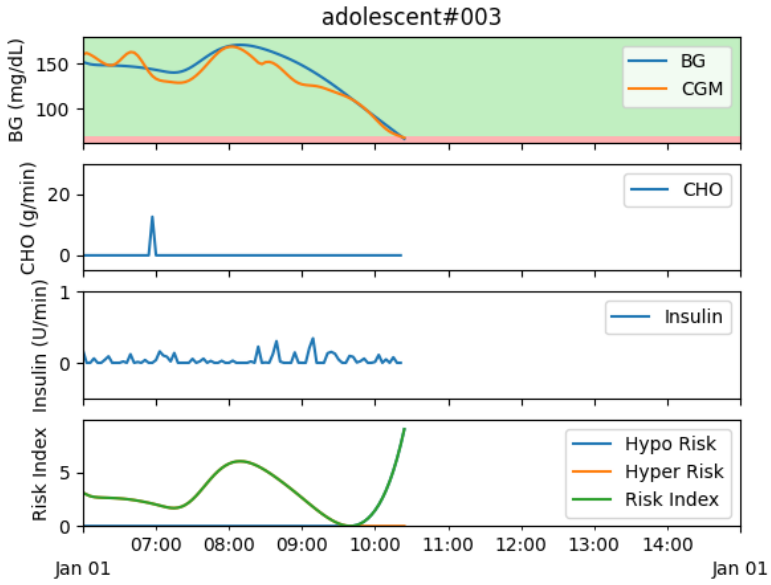
[10] https://stable-baselines3.readthedocs.io/en/v2.1.0/modules/a2c.html
[11] https://stable-baselines3.readthedocs.io/en/v2.1.0/modules/dqn.html
[12] https://stable-baselines3.readthedocs.io/en/v2.1.0/modules/ppo.html
[13] https://stable-baselines3.readthedocs.io/en/v2.1.0/modules/sac.html
[14] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html



Irudia 7.8: A2C algoritmoa erabiltzean lortzen diren emaitzak. Gehiegizko *intsulina* erabiltzen da eta beraz simulazioa ez du ezer irauten (balio arriskutsuetara heltzen da azkar).



Irudia 7.9: PPO algoritmoa erabiltzean lortzen diren emaitzak. Diabetiko heldu baten simulazioa. Nerabeek baino gehiago irauten dute.



PPO algoritmoa erabiltzean lortzen diren emaitzak.

Irudia 7.10: Diabetiko nerabe baten simulazioa. Modeloak *intsulina* kantitatea ezarri egiten du. *Gluzemia* oso baxuan bukatu egiten du, porrota.

5 eguneko entrenamenduaren emaitzak aztertu eta gero entrenamendu luzeago batekin probak egin dira. Konkretuki PPO eta DQN algoritmoak 10 egunez entrenatu dira. Zoritxarrez emaitzak ez dira asko aldatu, hobekuntza minimoa izan da. Gainera, gehiegizko *intsulina* kontuan hartu gabe, PPO-k hurrengo arazoak dituela ikusi da:

- Karbohidratoek (CHO) ez dute ageriko efekturik *intsulina* kalkuluan, eta beraz *gluzemia* altuak (200 mg/dl baino gehiago) ikusi daitezke (ikus 7.9. irudia). Hau algoritmoak zenbat jan den jasotzen ez duelako gertatzen dela dirudi. Karbohidratoak jaten badira agenteak *intsulina* kantitatea handitu beharko luke.
- *Gluzemia* baxua denean, algoritmoak ez du *intsulina* 0-ra murrizten, zerbait sartzen du beti.

Ikerketetan oinarritutako algoritmoaren emaitzak

Hasierako algoritmoak, PPO erabiliz, batzuetan simulazioa mantentzeko gai da. Hala ere ez da simulagailua etengabe mantentzeko gai eta gaixo mota batzuekin ez du ondoegi funtzionatzen, adibidez nerabeekin. Gainera beste teknika sinpleagoekin konparatuta, formula matematikoak adibidez, errendimendu txarragoa du. Ondorioz emaitza hobeak dituen algoritmoa bilatu da. Lee et al. [7] ikerketan

deskribatzen den algoritmoa erabili da.

Ikerketaren arabera pazienteen *intsulina* etengabe maneiatzeko eta % 89.56 denboran balio osasuntsuen artean mantentzeko gai da (80 eta 180 artean). Algoritmo honen Python inplementazioa idatzi eta entrenamendurako erabili da. Entrenamendua *SB3* algoritmoen denbora berdinekoa izan da, lehenengo 5 eguneko eta ondoren 10 eguneko entrenamendua egin da. Zoritxarrez algoritmoaren Python inplementazioaren emaitzak ez dira onak izan. Simulatutako gaixo batzuk mantentzeko gaitasuna du, baina ez denbora luzez. Hainbat proba egin eta gero ikerketan oinarritutako algoritmoaren bidez eta SB3-ren PPO bidez lortutako modeloaren arteko desberdintasuna oso txikia da.

Zoritxarrez ez dira erabilgarriak diren emaitzak lortu, ez dute formula matematikoen bidez lortzen dituen emaitzak hobetzen. Emaitza hauen arrazoia algoritmoaren Python inplementazioan erroren bat egotea izan daiteke. Hortaz aparte, irakurritako ikerketa gehienen algoritmoen ikasketan denbora luzea eman dela adierazten da. Ikasketa gailu azkarrago batzuk erabiliz edo denbora gehiago emanda emaitzak hobetzea posiblea izango litzatekeela uste da.

8 Probak

Atal honetan funtzionalitate berriaren inplementazioari egindako probak azalduko dira. Proba hauen bidez kodearen portaera zuzena ziurtatu eta hainbat errore aurkitu eta zuzendu dira. Proba hauek aplikazioa erabiliz eta JUnit[1] proben bidez egin dira.

8.1 Eskuzko probak

Funtzionalitatea inplementatu eta gero, aplikazioa erabiltzean errorerik ez dagoela konprobatu da. Honetarako funtzionalitatearen egoerak aldatzen dituzten elementuak kontuan hartu behar dira:

- **Aktibazio botoia**: funtzioa aktibatzeko eta desaktibatzeko botoia. *RL modeloa*ren bitartez kalkuluak egin eta hauek ikustaraztea kontrolatuko du.
- ***RL* modelo aukeraketa botoia**: *RL modeloa*ren fitxategia aukeratzeko botoia. Botoia inoiz sakatu ez bada, aplikazioak ez du modelo fitxategirik gordeta izango. Gainera fitxategiak erroreak izan ditzake, eta beraz fitxategia erabiltzen duen kodean salbuespenak sortu daitezke.

Elementuak kontuan hartuta, hurrengo portaerak espero dira:

1. **Funtzioa desaktibatuta egotean**: ez da *intsulina* gomendiorik kalkulatuko ezta hasierako interfazean agertuko (ikus 8.1. irudia)
2. **Funtzioa aktibatuta eta modelo hutsa edo erroreekin**: *xDrip*-en hasierako interfazean errore bat sortu dela adieraziko da (ikus 8.2. irudia)
3. **Funtzioa aktibatuta eta modeloa zuzena**: modeloak lortzen duen emaitza *xDrip*-en hasierako interfazean adieraziko da (ikus 8.3. irudia).

Inplementazioak portaera hauek betetzen dituela egiaztatu da (8.1, 8.2 eta 8.3 irudietan ikusten den bezala).

[1] Java kodearen probentzako sortutako liburutegia. Java kodea exekutatu eta aurretik idatzitako baieztapenen bidez emaitzak konprobatuko ditu.

Irudia 8.1: Funtzioa aktibaturik ez egotekotan ez dira *RL* emaitzak adieraziko.

Irudia 8.2: Funtzioa aktibatua baina modeloa erabiltzean errore bat gertatzen bada egongo den interfazea.

Irudia 8.3: Funtzioa aktibatua eta dena ondo badoa (emaitzan adierazten den *intsulina* beharra ez da zuzena, jostailuzko modeloa erabili da)

8.2 JUnit test-ak

JUnit test-ak kodea sakonean probatzeko sortu dira. Mockito[2] eta Robolectric[3] erabili dira probak egiteko. *xDrip*-en bi klase berri sortu eta probatu dira, "Calculations" eta "RLModel". Proba guztiak errorerik gabe exekutatzen dira eta %80-eko estaldura baino gehiago lortu da (ikus 8.4. irudia). Klase bakoitzean egindako probak azalduko dira, bakoitza bere atalean.

Aldaketak jaso dituzten *xDrip* klaseak erabilera proben bidez konprobatu dira bakarrik. Ez dituzte aurretik sortutako JUnit probarik, izan ere Android-en arazo tekniko batzuen ondorioz klase hauetan JUnit probak erabiltzea nahiko konplexua da.

Element	Class, %	Method, %	Line, %
Calculations	100% (2/2)	90% (9/10)	83% (47/56)
RLModel	100% (5/5)	100% (13/13)	100% (35/35)

Irudia 8.4: JUnit proben estaldura.

RLModel

"RLModel" klaseko probak laburki azalduko dira. Klasearen helburua *RL* modeloa erabiliz gomendioak kalkulatzea da. Hauek dira egindako probak:

- RLInput klasea: "RLModel" klasearen barruan definitzen den klasea da. Inferentzia metodoak erabilitako datuak gordetzeko sortu den klasea da. Bi test sortu dira klasea probatzeko:
  - **testCreateRLInput()**: klasearen eraikitzaileak ez duela errorerik sortzen konprobatzen du.
  - **testGetLatestBG()**: azken *gluzemia* balioa lortzeko metodoa probatzen du.
- **testConstructor()**: klasearen eraikitzailearen testa da. Bi aukera probatzen dira, *RL* modeloa egokia edo hutsa izatea.
- **testInferInsulin()**: *intsulina* beharra kalkulatzen duen metodoaren proba. Modeloa hainbat egoeratan probatzen da, metodoaren emaitza egoera guztietan zuzena dela ziurtatzeko. Modeloa ezegokia edo jasotako datuak "null" badira, metodoak "InferErrorException" salbuespena jaurti beharko du. *TF* liburutegiaren emaitzak Mockito bidez aurrezarri dira.

[2]Probak egitean klase baten portaera simulatzeko Framework-a. https://site.mockito.org/

[3]Probak egitean Android sistemaren elementuak simulatzeko erabili daitekeen Framework-a. https://robolectric.org/

Calculations

"Calculations" klaseko probak laburki azalduko dira. Klase honek orden bat jarraitu behar du (lehenengo modeloa inportatu, ondoren modeloa kargatu, eta bukatzeko modeloa erabili). Hurrengo probak egin dira (ordena jarraituz):

- **test_1_Import()**: modeloaren inportatze metodoari Android-eko URI bat emanez errorerik sortzen ez duela eta URI-aren fitxategia inportatu duela konprobatuko du.
- **test_2_LoadModel()**: inportatze proban sortutako fitxategia *RL* interpretatzaile bezala (*TF* modeloarekin lan egiteko Java klasea) kargatzen dela konprobatzen duen proba da. Modeloa "null", ez egokia edo zuzena dela probatuko da.
- **test_3_getRLInput()**: gomendioak egiteko behar diren datuak lortzeko metodoa probatzen du. Datu hauek *xDrip*-en datu-basetik lortzen dira. Datubasea hutsik eta datuak izatearen egoerak probatzen ditu.
- **test_4_calculateInsulin()**: modeloa erabiliz *intsulina* beharra kalkulatzen duen metodoa. Aurretik sortutako datuak erabiliz metodoak errorerik sortzen ez duela eta emaitza zuzena itzultzen duela konprobatzen du.

9 | Ondorioak

GrAL-aren bidez kode irekiko proiektu baten aldaketa eta ezaugarri berri baten inplementazioan esperientzia lortu da. Egindako lana egituratua eta jarraitua izan denez unibertsitateko beste zereginekin batera egin ahal izan da.

Kode irekiko proiektuetan garatu daitezkeen hainbat arazo aurkitu eta hauek zuzentzen ikasi da. Beste garatzaileen kodea irakurtzeko, ulertzeko eta aldatzeko gaitasunak garatu dira. Kode ezezagunaren arazoak identifikatu eta hauekin lan egitea ikasi da. Proiektu handi batean ezaugarri berri bat inplementatzea lortu da beste funtzionalitateak kaltetu gabe. Garatutako kodearen eskuzko probak baita JUnit probak garatu dira, kodearen kalitatea ziurtatuz.

Erreminta eta teknologia berriak erabili ez ezik, interes pertsonala dituzten teknologietan murgildu da proiektua, *Reinforcement Learning*-en adibidez.

Bukatzeko, esan beharra dago proiektuan zehar hainbat akats egin direla. Akats hauetatik ikasi da etorkizunean garatuko diren proiektuetan ez errepikatzeko.

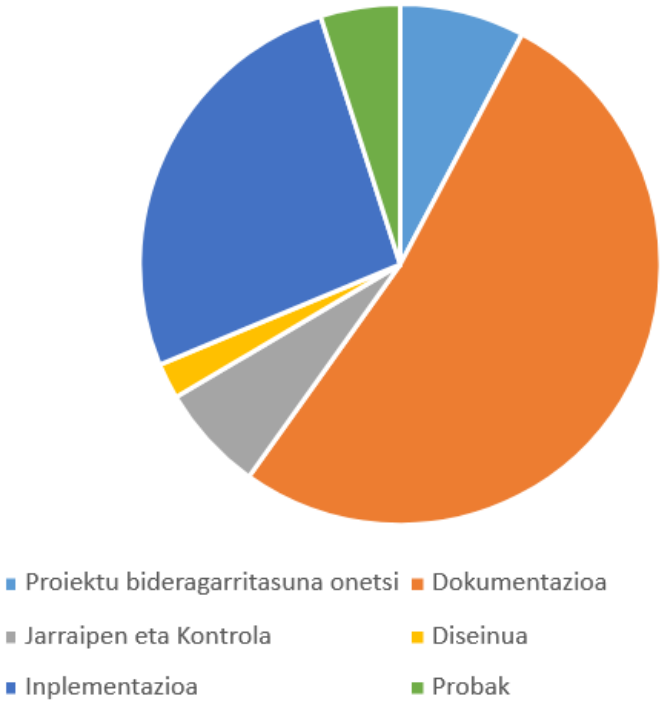
9.1 Planifikazio eta denbora errealen arteko desberdintasunak

GrAL-ean egindako lanaren planifikazioa ondo egitea nahiko zaila zen. Kode irekiko proiektuetan eta *RL*-en esperientzia gutxi izateak ziurgabetasun handia sortu du. Gainera *GrAL*-aren dokumentazioan idatzi beharreko edukia handitzen joan da. Ondorioz planifikatutako orduen eta sartutako ordu errealen artean desberdintasun handiak sortu dira.

9.1. irudian planifikazio eta denbora errealen arteko desberdintasunak ikusi daitezke. Ataza bakoitzeko bi barra ditugu, goikoa denboraldi erreala da eta behekoa planifikatua. Ataza aurreratu bada planifikazio barra berdea izango da eta gorria atzeratu bada. Barra bakarra egotekotan planifikazio bete da.

Agerikoa da dokumentazioan atzerapen asko egon dela. Dokumentazioak pla-

Irudia 9.1: Planifikazioen arteko desberdintasuna.



Irudia 9.2: *GrAL*-aren ataza bakoitzeko eman den denboraren grafikoa.

nifikazioan pentsatu baino eduki gehiago eta konplexuagoak izan ditu. Ondorioz denbora askoz gehiago eman behar izan da dokumentazioa idazten. Gainera, atzerapen hauen ondorioz *GrAL*-aren entrega eta aurkezpenak atzeratuak izan dira iraileko deialdira. Beste aldetik nabarmenekoa da funtzionalitatearen garapena, nahiz eta nahiko konplexua izan, aurrerapen handiak izan dituela.

9.2. irudiaren bidez ikusi daiteke *GrAL*-ean emandako orduen erdia dokumentazioa egiten eman direla. Beste aldetik funtzionalitatearen laurdena inplementazioa garatzen izan da. Bukatzeko beste ataza guztien konbinazioa proiektuaren laurdena izan da. Guztira 361 ordu erabili dira, espero baino 20 ordu gutxiago. Atzerapenak egunero eman beharreko orduak errespetatu ez direlako gertatua dira, arazo pertsonalen ondorioz.

9.2 Etorkizunerako hedapenak

Proiektua garatu eta gero argi geratu da *xDrip* aplikazioan eta diabetesaren maneiuaren arloan hobekuntza asko egin daitezkela. *xDrip*-ek, adibidez, kode hobekuntza handiak behar ez ezik, zuzendu beharreko *issue* asko ditu.

Beste aldetik, diabetesaren maneiuari buruzko hainbat paper irakurri eta gero argi dago *RL* modeloek diabetikoen bizitza hobetu dezaketela. Etorkizuneko *GrAL* batean *RL* modelo berri bat sortu daiteke, eta ondoren proiektu honetan inplementatutako sistemara txertatu. Ikerketa munduan modelo hauen interesa agerikoa da.

9.3 Balorazio pertsonala

Proiektuan sortutako arazoak zuzendu eta helburuak bete dira. Proiektu handietan eskua sartzeko eta beste garatzaileekin komunikatuz funtzionalitate berri bat garatzeko beldurra galdu dut.

Teknologia berrietan esperientzia lortu dut. Adibidez *RL* modeloa garatu dut eta bidean arlo horretan egin diren zenbait ikerketa ezagutu ditut. Beste aldetik, Android aplikazioen eta bereziki software irekien garapenean lortutako trebetasuna oso aberasgarria iruditu zait.

Acronyms

API *Application Programming Interface*. 42, 75

CGM *Continuous Glucose Monitor*. vi, 25, 26, 28, 29, 35, 36, 77, 84

EAO *Estatuko Aldizkari Ofiziala*. 22

GrAL Gradu Amarierako Lana. 23, 32, 45, 49, 55, 67, 93, 95, 96

Gym *OpenAI Gym*. 42, 55, 75, 77, 78

IDE *Integrated Development Environment*. 14–16, 21, 50

IP *Intsulina-Ponpa*. 29

MDP *Markov Decision Process*. 37–39, 41

ML *Machine Learning*. 36

OSS Open Source Software. 6, 9, 24, 49, 99

PPO *Proximal Policy Optimization*. 83

RL *Reinforcement Learning*. iii, vi, 1, 3–5, 9, 11, 21, 23, 24, 36–38, 40–45, 47, 49, 55–57, 60, 63–65, 67–69, 71, 72, 74, 75, 78, 81–84, 89–93, 96, 100

SB3 *Stable Baselines 3*. 42, 55, 75, 76, 78, 80, 84, 87

SG *Simglucose*. 41, 42, 60, 75, 76, 83, 84

TF *Tensor Flow*. 60, 78, 91, 92

Glosarioa

Application Programming Interface Aplikazio desberdinei elkarren artean komunikatzeko eta informazioa eta funtzionaltasunak partekatzeko aukera ematen dien kode-pieza da. 97, 100

commit Egindako aldaketak Git biltegi batera bidaltzen dituen operazioa. 3, 27

Continuous Glucose Monitor Pertsona baten *gluzemia* kalkulatzen duen gailua. Ikus 4.1.3. atala informazio gehiagorako. 35, 97

discussion GitHub proiektuek dituzten aukerazko atala (garatzaileek aktibatu edo desaktibatu dezakete). Proiektuari buruz hitz egiteko foru bezala lan egiten du. 52, 67

Estatuko Aldizkari Ofiziala Espainiar Estatuaren aldizkari ofiziala da. 22, 97

fork *OSS* baten pertsona batek egindako kopia pertsonala. 19, 76, 100

gluzemia Pertsona batek odolean duen glukosa kontzentrazioa. Normalean odoleko azukrea bezala ezagutzen da. 1, 2, 5, 25, 26, 28, 31–36, 42, 43, 45, 47, 52, 53, 55, 59, 64, 68, 71–73, 77, 81–84, 86, 91, 99, 100

Integrated Development Environment Software garapenerako erabiltzen den erreminta. 97

intsulina Pertsona guztiek sortzen duten hormona, Diabetes T1 duten pertsonak izan ezik. Diabetesa duten pertsonek intsulina behar dute bizitzeko. Bi motako intsulinak daude, basal eta bolo intsulina. Lehenengoa egun osoaren intsulina maila mantentzeko erabiltzen da eta bigarrena jan egiteko orduan sortutako *gluzemia* gorakada gelditzeko. Informazio gehiago lortzeko Oinarri teorikoak irakurri. iii, vi, 1, 2, 5, 6, 26, 29, 31–34, 41–45, 47, 52, 53, 55–57, 59, 60, 63, 64, 68, 69, 71–76, 81, 83–87, 89–92, 99

Intsulina-Ponpa *intsulina* gordetzeko gaitasuna duen eta kateter baten bidez pertsona baten organismoan *intsulina* etengabe administratu dezakeen gailua da.. 29, 97

issue GitHub proiektuek dituzten atala. Atal honetan erabiltzaileek dituzten arazoak garatzaileei jakinarazi dezakete baita funtzionalitate berriak eskatu. 3, 95

LaTeX Testuaren konpozizioa sortzeko erreminta. 21

Machine Learning Garatzaile baten ordez, ordenagailuak beraien algoritmoak "aurkitzen" ikastea garatzen duen informatikako eremua da.. 36, 97

Markov Decision Process *RL* atazen definizio matematikoa.. 37, 41, 97

modeloa Objektu, pertsona edo sistema bati buruz informazioa ematen duen deskribapena. Dokumentu honen kasuan *Reinforcement Learning*-en algoritmo entrenatuari buruz hitz egiteko erabiltzen da "modelo" hitza. 5, 9, 11, 41, 43, 44, 49, 55, 63, 67, 69, 71, 75, 80, 89

OpenAI Gym *RL* ingurune baten eta honetan erabili nahi den algoritmoa lotzeko *Application Programming Interface* ireki[1] bat da. 42, 75, 77, 97

Proximal Policy Optimization 2017-an OpenAI[2] enpresak garatutako *Reinforcement Learning* algoritmoa.. 83, 97

pull request Git projektu baten *fork*-an agindako aldaketak oinarrizko projektura igotzearen eskaera. 3, 24, 68

Reinforcement Learning Sarietan oinarritutako ikasketa automatikoa. iii, 1, 3, 5, 31, 36, 43, 93, 97, 100

Simglucose Diabetes duen pertsona baten *gluzemia* fluktuazioen eta insuluinaren efektuen simuladorea. Simglucose[3] Github proiektu orrian aurkitu ahal da. 42, 60, 75, 83, 97

Stable Baselines 3 *RL* algoritmoen inplementazioak dituen PyTorch-en oinarritutako liburutegia. 21, 42, 75, 84, 97

Tensor Flow Google-ek sortutako eta ikasketa automatikoan erabiltzen den kode irekiko liburutegia da. 60, 78, 97

xDrip Diabetes gaixotazuna maineilatzeko aplikazioa. 1–5, 15, 19, 24–29, 41, 43, 47–52, 54–57, 60, 63, 64, 67, 69, 89, 91, 92, 95

[1] https://github.com/openai/gym
[2] https://openai.com/
[3] https://github.com/jxx123/simglucose

Bibliografia

[1] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, Aug 2021.

[2] Taiyu Zhu, Kezhi Li, Lei Kuang, Pau Herrero, and Pantelis Georgiou. An insulin bolus advisor for type 1 diabetes using deep reinforcement learning. *Sensors*, 20(18), 2020.

[3] Qingnan Sun, Marko V. Jankovic, João Budzinski, Brett Moore, Peter Diem, Christoph Stettler, and Stavroula G. Mougiakakou. A dual mode adaptive basal-bolus advisor based on reinforcement learning. *IEEE Journal of Biomedical and Health Informatics*, 23(6):2633–2641, 2019.

[4] Global Burden of Disease CollaborativeÑetwork. Global burden of disease study 2019. Institute for Health Metrics and Evaluation, 2020. https://vizhub.healthdata.org/gbd-results/.

[5] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The UVA/PADOVA type 1 diabetes simulator: New features. *J Diabetes Sci Technol*, 8(1):26–34, January 2014.

[6] Jinyu Xie. Simglucose v0.2.1 (2018).

[7] Seunghyun Lee, Jiwon Kim, Sung Woon Park, Sang-Man Jin, and Sung-Min Park. Toward a fully automated artificial pancreas system using a bioinspired reinforcement learning design: In silico validation. *IEEE Journal of Biomedical and Health Informatics*, 25(2):536–546, 2021.

[8] Chirath Hettiarachchi, Nicolo Malagutti, Christopher Nolan, Eleni Daskalaki, and Hanna Suominen. A reinforcement learning based system for blood glucose control without carbohydrate estimation in type 1 diabetes: In silico validation. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 950–956, 2022.

[9] Kok-Lim Alvin Yau, Yung-Wey Chong, Xiumei Fan, Celimuge Wu, Yasir Saleem, and Phei-Ching Lim. Reinforcement learning models and algorithms for diabetes management. *IEEE Access*, 11:28391–28415, 2023.

[10] E.D. Lehmann, T. Deutsch, E.R. Carson, and P.H. Sönksen. Aida: an interactive diabetes advisor. *Computer Methods and Programs in Biomedicine*, 41(3):183–203, 1994.

[11] RichardÑ. Bergman. Minimal Model: Perspective from 2005. *Hormone Research*, 64(Suppl. 3):8–15, 01 2006.

[12] Pasquale Palumbo, Simona Panunzi, and Andrea De Gaetano. Qualitative behavior of a family of delay-differential models of the glucose-insulin system. *Discrete and Continuous Dynamical Systems - B*, 7(2):399–424, 2007.

[13] Harry Emerson, Matthew Guy, and Ryan McConville. Offline reinforcement learning for safer blood glucose control in people with type 1 diabetes. *Journal of Biomedical Informatics*, 142:104376, 2023.

[14] Philip Virgil Astillo, Gaurav Choudhary, Daniel Gerbi Duguma, Jiyoon Kim, and Ilsun You. Trmaps: Trust management in specification-based misbehavior detection system for imd-enabled artificial pancreas system. *IEEE Journal of Biomedical and Health Informatics*, 25(10):3763–3775, 2021.