Software Ingenieritza

Bomberman Sprint 1

Dokumentazioa

Asier Barrio, Asier Las Hayas, Gaizka Divasson eta Mikel Eguia

2025ko Martxoak 21

Aurkibidea

1 Deskribapena

Atal honetan klase bakoitzean erabilitako metodoak azalduko dira, eta nola integratu ditugun MVC eta Observer gure proiektuan.

1.1 Bloke klasea

1.1.1 Atributuak

- **private boolean dago:** *dago* aldagai boolearra da, blokeak jokoan jarraitzen duen edo suntsitua izan den adierazten duena. Haren balioa true bada, blokea oraindik presente dagoela esan nahi du; falsea bada, berriz, ezabatu egin dela adierazten du, normalean leherketa baten edo jokoan beste gertaeraren baten ondorioz.
- **private int posX eta private int posY:** Blokearen koordenatuak jokoaren matrizean irudikatzen dituzte.

  –
- **private boolean dago:** Blokeak jokoan jarraitzen duen (true) edo desagertu den (false lehertu ondoren) adierazten du.

1.1.2 Eraikitzailea

Klasearen eraikitzaileak hasten du bloke bat matrizearen barruko posizio espezifiko batean. Blokearen X eta Y koordenatuak irudikatzen dituzten bi parametro oso jasotzen ditu. Gainera, true a dago esleitzen du, eta blokea sortzean aktibo dagoela adierazten du.

```
1     public Bloke(int pPosX, int pPosY ){
2                 this.posX = pPosX;
3                 this.posY = pPosY;
4                 this.setDago(true);
5         }
```

1.1.3 badago() metodoa

Balio boolearra itzultzen du, blokeak jokoan jarraitzen duen adieraziz. Barruan, *getDago()* deitzen du, eta, horren ondorioz, metodo horrek ezizen deskribatzaileago gisa funtzionatzen du blokea agertokian dagoela egiaztatzeko, Dago aldagaira zuzenean sartu gabe.

```
1     public boolean badago() { //Jakiteko jokuan jarraitzen duen ala ez
2                 return getDago();
3         }
```

1.1.4 getPosizioa() metodoa

Metodo honek blokearen egungo koordenatuak irudikatzen dituzten bi elementu osoko array bat itzultzen du jokoaren matrizean. Blokea edozein unetan non dagoen jakiteko aukera ematen du, eta baliagarria da jokoaren barruan talkak, leherketak edo mugimenduak kalkulatzeko.

```
1    public int[] getPosizioa() {
2                 return new int[] {posX,posY};
3        }
```

1.1.5 getPosizioa() metodoa

Metodo horri esker, blokeak jokoaren matrizean duen egungo posizioa lor daiteke. Posizioa *posX* eta *posY* aldagaietan biltegiratzen da. Aldagai horiek blokearen X eta Y koordenatuak irudikatzen dituzte maparen egituran. Metodoa aipatzen denean, bi koordenatuak dituen osoko array bat itzultzen du metodoak *posX*, *posY*. Hori baliagarria da jolasa garatzeko, aukera ematen baitu bloke bakoitza eszenatokian zehazki non dagoen jakiteko eta hura manipulatzea errazten baitu, hala nola, hura mugitzea, talkak egiaztatzea edo bloke bat leherketa batek harrapatu duen zehaztea. Koordenatuak array gisa itzultzean, erraztu egiten da jokoaren beste funtzio batzuen barruan erabiltzea, eta saihestu egiten da bi metodo desberdin erabili behar izatea, bakoitza bere aldetik, *posX* eta *posY* lortzeko.

```
1    public int[] getPosizioa() {
2                 return new int[] {posX,posY};
3        }
```

1.1.6 getDago() metodoa

dago aldagaiaren egoerara itzultzen den sarbide-metodo bat da. Atributu horren balioa lortzeko modu kontrolatu bat ematea du helburu, kapsulatze-printzipioa ziurtatuz. Blokeak jokoan jarraitzen duen edo suntsitu den zehazteko erabiltzen da.

```
1    public boolean getDago() {
2                 return dago;
3        }
```

1.1.7 setDago() metodoa

Blokearen egoera aldatzea ahalbidetzen duen metodo aldatzailea. Parametro boolearra jasotzen du eta *dago*-ri esleitzen dio. Blokea suntsitu behar denean erabiltzen da, haren balioa faltsutu ordez. Bloke bat birsortzeko ere erabil liteke, jokoan beharrezkoa balitz.

```
1       public void setDago(boolean dago) {
2                   this.dago = dago;
3           }
```

1.2 BlokeZerrenda klasea

1.2.1 Atributuak

- **private `ArrayList<Bloke>` blokeZerrenda:** Atributu honek Bloke klaseko objektuen zerrenda bat da, ArrayList datu-egitura batean biltegiratuak. Bere funtzio nagusia jokoaren barruan bloke multzo bat kudeatzea da, berariazko metodoen bidez manipulatzeko aukera emanez. ArrayList hautatzeak blokeak modu eraginkorrean gehitzea, ezabatzea eta eskuratzea errazten du. Zerrenda pribatua denez, BlokeZerrenda klasearen barruan baino ezin da manipulatu, datuen egituraren gaineko kontrol egokia ziurtatuz. Horrek aukera ematen du blokeek jokoaren mapan duten kokapena kudeatzeko, leherketak erregistratzeko eta blokeetara zerrendan duten posizioaren arabera sartzeko.

1.2.2 Eraikitzailea

Eraikitzaile honek *blokeZerrenda* atributua abiarazten du `ArrayList <Bloke>` instantzia berri gisa, zerrenda huts bat sortuz, non jokoaren blokeak gordeko diren. Bere helburua da BlokeZerrenda objektu bakoitza erabiltzeko prest dagoen datu-egitura batekin hastea, erreferentzia-errore nuluak saihestuz. Eraikitzaile hori aipatzean, blokeak zerrendara berehala gehi daitezkeela bermatzen da, eskuz hasieratu beharrik gabe. BlokeZerrendaren instantziak sortzeko funtsezko urratsa da, jokoaren barruan blokeak kudeatzeko oinarria ezartzen baitu.

```
1     public BlokeZerrenda() {
2                 this.blokeZerrenda= new ArrayList<>();
3         }
```

1.2.3 getIter() metodoa

Metodo honek iteratzaile bat itzultzen du bloke-zerrenda (*blokeZerrenda*) zeharkatzeko. Haren erabilera nagusia zerrendako elementuen gainean iteratzeko modu eraginkorra ematea da, azpiko datu-egitura zuzenean azaldu gabe. Metodo pribatua da, eta horrek esan nahi du BlokeZerrenda klasean bakarrik erabil daitekeela. Iterator erabiltzean, elementuak errazago ezabatzen dira iterazioan, errorerik sortu gabe. Metodo hau erabilgarria da zerrenda zeharkatu behar duten eragiketetan, hala nola blokeak inprimatzean edo blokeak posizio jakin batean bilatzean.

```
1     private Iterator<Bloke> getIter(){
2                 return this.blokeZerrenda.iterator();
3         }
```

1.2.4 gehituBloke(Bloke pBloke) metodoa

Metodo honek *blokeZerrenda* zerrendari bloke berri bat eransteko aukera ematen du. Bloke motako objektu bat jasotzen du parametro gisa, eta ArrayList sis-

temara gehitzen du *add()* metodoaren bidez. Helburua da jokoan blokeak modu dinamikoan sartzea kudeatzea, eta aukera ematea jokaleku berriak sortzeko edo partida batean zehar elementuak gehitzeko. Funtsezko metodoa da joko-blokeak administratzeko; izan ere, metodo hori gabe, ezingo litzateke biltegiratutako bloke-multzoa aldatu. Zerrenda mapan dauden blokeekin eguneratuta mantenduko dela bermatzen du.

```
1    public void gehituBloke(Bloke pBloke) {
2                blokeZerrenda.add(pBloke);
3        }
```

1.2.5 deleteBloke(Bloke pBloke) metodoa

Metodo honek *blokeZerrenda*ren bloke espezifiko bat ezabatzen du. Bloke motako objektu bat jasotzen du parametro gisa, eta *remove()* bidez ezabatzen du. Bloke-zerrenda eguneratu nahi du elementu bat jolasetik desagertzen denean, adibidez, leherketa baten ondoren. Funtsezkoa da jokoaren egoeraren sendotasunari eusteko, suntsitutako blokeek zerrendan jarraitzen ez dutela ziurtatuz. Behar bezala funtzionatzeko, ezabatu beharreko blokea zerrendan egon behar da; bestela, ez du eraginik izango. Metodo hori funtsezkoa da agertokiko blokeen manipulazio dinamikoan.

```
1    public void deleteBloke(Bloke pBloke) {
2                blokeZerrenda.remove(pBloke);
3        }
```

1.2.6 getBloke(int index) metodoa

Metodo honek zerrendako bloke espezifiko bat itzultzen du parametro gisa emandako indizearen arabera. Blokea itzuli aurretik, egiaztatu indizea baliozkoa dela (> = 0 eta ¡ *blokeZerrenda.size ()*), erroreak saihesteko zerrendan ez dauden posizioetara sartzean. Aurkibidea zuzena bada, blokea posizio horretan itzultzen da; bestela, null itzultzen du. Bere funtzioa zerrendan gordetako blokeetara zuzenean sartzeko aukera ematea da, jokoaren barruan horiek manipulatzea eta kontsultatzea erraztuz. Erabilgarria da bloke espezifiko bat lortu behar denean zerrenda osoa eskuz egin gabe.

```
1    public Bloke getBloke(int index) {
2                if(index >= 0 && index< blokeZerrenda.size()) {
3                        return blokeZerrenda.get(index);
4                }
5                return null;
6        }
```

1.2.7 getBloke(int x, int y) metodoa

Bilaten du bloke bat (x, y) koordenatuetan, eta, Biguna instantzia bat bada, dei ezazu *eztandaEgin()* metodoa, jokotik kenduz. Horri esker, leherketa batean

bloke hauskorren suntsiketa simulatu daiteke, jolasaren inguruneari modu dinamikoan eraginez.

```
1     public Bloke getBloke(int X, int Y) {
2                 Iterator<Bloke> iter=getIter();
3                 Bloke pBloke = null;
4                 boolean aurkitua = false;
5                 while(iter.hasNext() && !aurkitua) {
6                         pBloke=iter.next();
7                         if (pBloke.getPosizioa()[0]== X && pBloke.getPosizioa()[1]== Y &&
                              pBloke != null) {
8                                 aurkitua = true;
9                         }
10                }
11                return pBloke;
12        }
```

1.2.8 getBlokeGuztiak() metodoa

Metodo honek *blokeZerrenda*n gordetako blokeen zerrendaren kopia bat itzultzen du. Jatorrizko zerrendan zuzeneko aldaketak saihesteko, ArrayList instantzia berri bat sortzen da *blokeZerrenda*ren edukiarekin. Helburua da jokoan dauden bloke guztietarako sarbidea ematea, barne-zerrendaren osotasuna arriskuan jarri gabe. Horri esker, beste klase edo metodo batzuek blokeekin lan egiten dute, ustekabeko alterazioak izateko arriskurik gabe. Erabilgarria da bloke-bilduma osoa prozesatu behar denean, jatorrizko egituratik kanpo aldaketarik egin gabe.

```
1     public ArrayList<Bloke> getBlokeGuztiak(){
2                 return new ArrayList<>(blokeZerrenda);
3         }
```

1.2.9 eztandaEginPosizioan(int x, int y) metodoa

Metodo honek (x, y) posizioan dagoen bloke bat bilatzen du zerrendan, eta, aurkitzen badu, *eztandaEgin()* metodoa aktibatzen du, leherketa eraginez. Iteratzaile bat (*Iterator<Bloke>*) erabiltzen du zerrendan zehar ibiltzeko eta bloke bakoitzaren koordenatuak proportzionatuekin alderatzeko. Bloke bat adierazitako posizioan aurkitzen baduzu, lehertu egiten da eta *break* bidez bilaketa gelditzen du, beharrezkoak ez diren iterazioak saihestuz. Metodo hori funtsezkoa da jokoaren mekanikan, bloke jakin baten detonazioa jokalariaren ekintzen edo kateko leherketen arabera simulatzeko aukera ematen baitu.

```
1     public void eztandaEginPosizioan(int x, int y) {
2                 Iterator<Bloke> iter=getIter();
3                 while(iter.hasNext()) {
4                         Bloke pBloke=iter.next();
5                         if (pBloke != null) {
```

```
                                if (pBloke.getPosizioa()[0] == x && pBloke.getPosizioa()
                                    [1] == y && (pBloke instanceof Biguna)) {
                                        ((Biguna)pBloke).eztandaEgin();
                                        break;
                                }
                        }
                }
        }
```

1.2.10 blokeakInprimatu() metodoa

Metodo horrek *blokeZerrenda*n gordetako bloke guztien informazioa inprimatzen du kontsolan. Iteratzaile bat (*Iterator<Bloke>*) erabiltzen du zerrenda zeharkatzeko eta bloke bakoitzaren atributuak lortzeko: mota (*mota*), matrizean duen posizioa (*posX*, *posY*) eta existitzen jarraitzen duen (*dago*). Gero, inprimatu balio horiek formatu irakurgarrian. Bere funtzioa jolaseko blokeen egungo egoeraren testu-irudikapena ematea da, denbora errealean arazteko edo monitorizatzeko baliagarria izan daitekeena. Blokeen antolaera eta propietateak grafikoen beharrik gabe bistaratzea errazten du.

```
    public void blokeakInprimatu() {
                Iterator<Bloke> iter=getIter();
                while(iter.hasNext()) {
                        Bloke bloke=iter.next();
                        System.out.println("Posizioa:␣("+bloke.getPosizioa()[0]+ ",␣"+
                            bloke.getPosizioa()[1]+")␣Dago"+bloke.badago());
                }
        }
```

1.2.11 blokeBigunaDu() metodoa

Egiaztatu zerrendak gutxienez *biguna* motako bloke bat duen. Bat aurkitzen badu, true itzultzen da; bestela, false itzultzen du. Baliagarria da mapan oraindik bloke hauskorrik dagoen egiaztatzeko, eta horrek partidaren logikan eragina izan dezake.

```
    public boolean blokeBigunaDu() {
                Iterator<Bloke> iter=getIter();
                boolean dauka = false;
                while(iter.hasNext() && !dauka) {
                        Bloke pBloke=iter.next();
                        if (pBloke instanceof Biguna) {
                                dauka = true;
                        }
                }
                return dauka;
        }
```

1.3 Biguna klasea

1.3.1 Atributuak

- **private boolean eztanda:** Blokea leher daitekeen adierazten duen aldagai boolearra da. Hasieran true moduan ezartzen da, eta horrek iradokitzen du blokea suntsitzeko prest dagoela. Ikasgelan zuzenean erabiltzen ez bada ere, leherketa baldintzatzeko edo etorkizunean jokoa handitzeko balio dezake.

1.3.2 Eraikitzailea

Biguna klaseko eraikitzailea, Bloke oinordetzan hartzen duena. Koordenatuak jasotzen ditu (*pPosX, pPosY*) eta superklasearen eraikitzaileari pasatzen dizkio. Blokea jolasaren barruan zehaztutako posizioan hasten du, eta leherketa batek suntsi dezakeen bloke gisa markatzen du.

```
1     public Biguna(int pPosX, int pPosY) {
2                 super(pPosX, pPosY);
3
4         }
```

1.3.3 eztandaEgin() metodoa

Blokearen leherketa simulatzen du, kontsolan mezu bat inprimatuz eta *setDago (false)* deituz. Horrek adierazten du blokea suntsitu egin dela. Metodo hori funtsezkoa da leherketen mekanika inplementatzeko, jokoaren blokea ezabatuz bere posizioan detonazio bat gertatzen denean.

```
1     public void eztandaEgin() {
2                 System.out.println("Blokea␣eztanda␣egin␣du.");
3                 setDago(false);
4
5         }
```

1.4 Gogorra klasea

1.4.1 Eraikitzailea

Gogorra klaseko eraikitzailea, Bloke oinordetzan hartzen duena. Koordenatuak jasotzen ditu (*pPosX, pPosY*) eta superklasearen eraikitzaileari pasatzen dizkio. Atributu eta metodo berririk zehazten ez denez, bloke mota honek Bloke ezaugarri guztiak mantentzen ditu, baina jokoan bloke suntsiezin bat adieraz dezake.

```
1     public Gogorra(int pPosX, int pPosY) {
2                 super(pPosX, pPosY);
3
4         }
```

1.5 Bonba klasea

1.5.1 Atributuak

- **private int radioa :** Bonbaren leherketa-erradioa irudikatzen du. Balio horrek zehazten du zein distantziataraino eragin dezakeen detonazioak jatorri-puntutik. Balio oso bat da, eraikitzailean jasotako balioarekin abiarazten dena, eta ez da aldatzen programa exekutatzen den bitartean. Bonbak eztanda egiten duenean, talkaren irismena zehaztea du helburu.
- **private int X :** Bonbak labirintoaren barruan duen posizio horizontala adierazten du. Eraikitzailean esleitzen den balio oso bat da, eta bonba lehertuko den kokapen zehatza ezagutzeko aukera ematen du. Atributu hori funtsezkoa da leherketa gertatzen denean labirintoko zein gelaxka edo blokeri eragingo zaion zehazteko.
- **private int Y :** Bonbak labirintoaren barruan duen posizio bertikala adierazten du. *X* bezala, eraikitzailean esleitzen da eta eztandaren inpaktu-puntua kalkulatzeko erabiltzen da. *X*-rekin batera, balio horrek bonba leherrarazi ondoren labirintoaren zein zati aldatuko den identifikatzen du.
- **private boolean eztanda:** Aldagai boolearra da, eta bonba lehertzeko aktibo dagoen adierazten du. "True" gisa hasieratzen da, eta horrek esan nahi du denbora jakin baten ondoren lehertzeko prest dagoela. *UpdateKont()* metodoaren bitartez aldatzen da haren balioa, eta, horri esker, eztandaren une zehatza kontrola daiteke.
- **private int kont:** Bonbak eztanda egin aurretik geratzen den denbora kudeatzeko erabiltzen den kontagailu baten moduan jarduten du. Hasiera batean, *PERIODO*rekin berdintzen da, eta *updateKont()* murriztu egiten da. Zerora iristen denean, bonbak kontagailuaren balioa lehertzen eta berrabiarazten du, eta portaera ziklikoa ziurtatzen du objektua berrerabiltzen bada.
- **private int PERIODO:** Konstante oso bat da, bonbak eztanda egin aurretik zenbat denbora-ziklo dauden adierazten duena. Haren balioa 2 da, eta horrek adierazten du bonba tenporizadorearen bi iterazioren ondoren lehertuko dela. Balio horrek detonazioaren aurreko denbora-tartea zehazten du.
- **private Timer timer:** *Timer* klaseko objektu bat da, eztandarako erregresio-kontua kudeatzeaz arduratzen dena. Eraikitzailean hasten da, eta aldizkako zeregin bat egiten du (*TimerTask*), *updateKont()* metodoari deitzen diona, bonbak zehaztutako denboraren ondoren eztanda egiten duela ziurtatuz.

1.5.2 Eraikitzailea

Bonba(int pRadioa, int pX, int pY) klasearen eraikitzailea da, eta bonbaren atributuak hasieratzeaz arduratzen da. Hiru parametro jasotzen ditu:

leherketa-erradioa (*pRadioa*), X posizioa (pX) eta Y posizioa (pY). Gainera, *TimerTask* bat sortzen du, *updateKont()* metodoa aldizka exekutatzen duena, ponpak aurrez definitutako aldi baten ondoren eztanda egiten duela ziurtatuz.

```
    public Bonba(int pRadioa, int pX, int pY){
                this.radioa = pRadioa;
                this.X = pX;
                this.Y = pY;
                eztanda = true;
                kont = PERIODO;
                TimerTask timerTask = new TimerTask() {
                        @Override
                        public void run() {
                                updateKont();
                        }
                };
                timer = new Timer();
                timer.scheduleAtFixedRate(timerTask, 0, 1000);
        }
```

1.5.3 updateKont() metodoa

TimerTask-aren exekuzio bakoitzean kont balioa batean murrizten duen metodo pribatua da. Kont zerora iristen denean, *PERIODO* balioa berrezartzen du, eztanda true aldatzen du, *detonar()* metodoari deitzen dio eta tenporizadorea gelditzen du. Era berean, bonbaren egoeraren aldaketari buruzko behatzaileak jakinarazteko diseinatuta zegoen.

```
    private void updateKont() {
                kont--;
                if (kont == 0) {
                        kont = PERIODO;
                        eztanda = true;
                        detonar();
                        timer.cancel();
                }
```

1.5.4 eztandaEgin() metodoa

Metodo honek, bonba eskuz leherrarazteko erabiltzen den metodoa da, gutxieneko denbora bat igaro ondoren. Metodoak, bonbaren radio barnean dauden gelaxkak lehertzen ditu eta blokeak desagerazteko *deleteBonba()* metodoari deitzen dio.

```
    public void eztandaEgin() {
                if (/*tenp>=3*/true){
                        int eztanda = this.radioa;
                        deleteBonba();
```

```
                }

        }
```

1.5.5 deleteBonba() metodoa

deleteBonba metodoa, jarritako bonba desagerrarazteko erabiltzen da. Pribatua da eta eztandaEgin() metodoaren bitartez egiten zaio deia.

```
    private void deleteBonba() {


        }
```

1.5.6 getRadioa() metodoa

Bonbaren erradioaren balioa itzultzen duen metodo pribatua. Honen bitartez zehazten da inguruko zenbat gelaxka lehertuko diren.

```
    private int getRadioa() {
                return this.radioa;
        }
```

1.5.7 apurtu() metodoa

Bonbaren leherketa gauzatzen duen metodoa da. Bloke bigunak apurtzea kudeatzen du. LaberintoEredua klasearen bitartez exekutatzen da.

```
private void apurtu() {
                    LaberintoEredua.getLabEredua().apurtuBlokeak(X,Y);
        }
```

1.5.8 getX() metodoa

Bonbaren X koordenatuaren balioa itzultzen duen metodo publikoa. Baliagarria da labirintoaren barruan duen posizioa lortzeko, atributura zuzenean iritsi gabe.

```
    public int getX() {
                return this.X;
        }
```

1.5.9 getY() metodoa

Bonbaren Y koordenatua itzultzen duen metodo publikoa. *GetX()* bezala, ponparen kokapena ezagutzeko aukera ematen du, haren atributuak zuzenean aldatu gabe.

```
    public int getY() {
                return this.Y;
        }
```

1.6 Gelaxka klasea

1.6.1 Atributuak

- **private static final long serialVersionUID:** Gelaxka klasea serializatzeko erabiltzen den long motako konstantea da. Serializatzen eta deserializatzen denean, klasearen bertsioa modu bakarrean identifikatzea da haren funtzioa. Hori garrantzitsua da bateragarritasun-akatsak saihesteko, klasea etorkizuneko bertsioetan aldatzen bada.
- **private GelaxkaEredu gelEredu:** *GelaxkaEredu* motako atributu pribatu bat da, labirintoaren barruko gelaxkaren eredua irudikatzen duena. Ikusmenaren (*Gelaxka*) eta jokoaren logikaren arteko lotura gisa jokatzen du, eta gelaxka automatikoki eguneratzen da eredua aldatzen denean.
- **private Image argazki:** *Image* motako atributua da, gelaxkari lotutako irudia motaren arabera gordetzen duena. Gelaxkaren egoeraren arabera (*gelEredu.getMota ()*), irudi espezifiko bat esleitzen da, hala nola bloke solido bat, bloke bigun bat, bonba bat edo pertsonaia nagusia.

1.6.2 Eraikitzailea

Gelaxka(GelaxkaEredu pGelEredu): *Gelaxka* klaseko eraikitzailea da. *GelaxkaEredu*ko instantzia bat jasotzen du parametro gisa, *gelEreduari* esleitzen dio eta behatzaile gisa erregistratzen du. Gainera, *ComponentAdapter* bat gehitzen du, osagaiaren tamainan aldaketak detektatzen dituena eta, ondorioz, irudia doitzen duena. Azkenik, *egokitu()* metodoa eguneratzeko deia egiten du, hasierako irudia esleitzeko.

```
    public Gelaxka(GelaxkaEredu pGelEredu) {
                this.gelEredu = pGelEredu;
                this.gelEredu.addObserver(this);
            addComponentListener(new ComponentAdapter() {
                @Override
                public void componentResized(ComponentEvent e) {
                        dimentsionatu();
                    }
                });
            egokitu();
        }
```

1.6.3 eguneratu() metodoa

Gelaxkaren irudia *gelEredu*-n biltegiratutako motaren arabera eguneratzen duen metodo pribatua da. Erabili switch egitura bat aurrez definitutako balioetan (1etik 12ra) oinarritutako irudi zuzena hautatzeko. Irudi egokia esleitu ondoren, *egokitu()*-ra deitzen dio , behar bezala eskalatzeko.

```
1  private void eguneratu() {
2
3  switch (gelEredu.getMota()) {
4
5  case 1:
6
7  argazki = (new ImageIcon(getClass().getResource("BlokeGogorra.png")).getImage());
8
9  break;
10
11 case 2:
12
13 argazki = (new ImageIcon(getClass().getResource("BlokeBiguna.png")).getImage());
14
15 break;
16
17 case 3:
18
19 argazki = (new ImageIcon(getClass().getResource("Bonba.png")).getImage());
20
21 break;
22
23 case 4:
24
25 argazki = (new ImageIcon(getClass().getResource("Eztanda.png")).getImage());
26
27 break;
28
29 case 5:
30
31 argazki = (new ImageIcon(getClass().getResource("Bonberman.png")).getImage());
32
33 break;
34
35 case 6:
36
37 argazki = (new ImageIcon(getClass().getResource("Ezker1.png")).getImage());
38
39 break;
40
41 case 7:
42
43 argazki = (new ImageIcon(getClass().getResource("Ezker2.png")).getImage());
```

```
44
45 break;
46
47 case 8:
48
49 argazki = (new ImageIcon(getClass().getResource("Eskuin1.png")).getImage());
50
51 break;
52
53 case 9:
54
55 argazki = (new ImageIcon(getClass().getResource("Eskuin2.png")).getImage());
56
57 break;
58
59 case 10:
60
61 argazki = (new ImageIcon(getClass().getResource("Behera1.png")).getImage());
62
63 break;
64
65 case 11:
66
67 argazki = (new ImageIcon(getClass().getResource("Behera2.png")).getImage());
68
69 break;
70
71 case 12:
72
73 argazki = (new ImageIcon(getClass().getResource("Gora1.png")).getImage());
74
75 break;
76
77 case 13:
78
79 argazki = (new ImageIcon(getClass().getResource("Gora2.png")).getImage());
80
81 break;
82
83 case 14:
84
85 argazki = (new ImageIcon(getClass().getResource("JokSutan.png")).getImage());
86
87 break;
88
89 case 15:
90
91 argazki = (new ImageIcon(getClass().getResource("BonbaEskuan.png")).getImage());
92
93 break;
```

```
94
95 default:
96
97 argazki = null;
98
99 setIcon(null);
100
101 break;
102
103 }
104
105 egokitu();
106
107 }
```

1.6.4 egokitu() metodoa

Metodo pribatu bat da, gelaxkaren irudia birdimentsionatzen duena osagaiaren beharrezko tamainara egokitzeko. Lehenik eta behin, egiaztatzen du argazkia ez dela null eta osagaiaren tamaina baliozkoa dela irudia *getScaledInstance()* metodoa erabiliz. Ondoren, eskalada-irudia ezartzen du JLabelen ikono gisa.

```
1 private void egokitu() {
2
3 if (!(argazki == null) && !((getWidth() <= 0 || getHeight() <= 0))) {
4
5 Image imagenEscalada = argazki.getScaledInstance(getWidth(), getHeight(), Image.);
6
7 setIcon(new ImageIcon(imagenEscalada));
```

1.6.5 update(Observable o, Object arg) metodoa

Observer interfazearen metodo inplementatua da. Automatikoki exekutatzen da modeloak (*GelaxkaEredu*) aldaketa bat jakinarazten duenean. Bere ekintza bakarra *eguneratu()* metodoa deitzea da; horri esker, gelaxkaren irudia dinamikoki eguneratzen da jokoaren egoeraren aldaketei erantzuteko.

```
1     public void update(Observable o, Object arg) {
2
3                  eguneratu();
4         }
```

1.7 GelaxkaEredu klasea

1.7.1 Atributuak

- **private int mota:** "int" motako atributu pribatu bat da, labirintoko gelaxka mota adierazten duena. Bere balioak gelaxkaren edukia definitzen

du, hala nola hormak, bonbak, sua edo pertsonaiak. Eraikitzailean hasten da eta *setTipo()* metodoarekin alda daiteke, behatzaileei aldatuz gero jakinaraziz.

1.7.2 Eraikitzailea

Klasearen eraikitzaileak *pMota* parametro bat jasotzen du, motari esleitzen diona. Balio horrek gelaxkaren hasierako egoera adierazten du labirintoan. Sortutako instantziak jolasaren objektu mota desberdinak irudika ditzake, eta ikusleen bidez identifikatu eta eguneratzeko aukera ematen du.

```
1    public GelaxkaEredu(int pMota) {
2        this.mota = pMota;
3    }
```

1.7.3 getMota() metodoa

Mota atributuaren balio eguneratua itzultzen duen metodo publikoa da. *Gelaxka* mota lortzeko erabiltzen da, egoera aldatu gabe. *Gelaxka* klasean erabilgarria da, labirintoaren barruko gelaxkaren edukiaren arabera zer irudi erakutsi behar den zehazteko.

```
1    public int getMota() {
2        return mota;
3    }
```

1.7.4 setMota(int nuevoTipo) metodoa

Metodo publiko bat da, eta mota-balioa parametro gisa jasotako mota berriarekin eguneratzen du. Deitzen du *setChanged()* eta *notifyObservers()* atalera, behatzaileei jakinarazteko gelaxka aldatu egin dela, interfaze grafikoak egoera berria isla dezan, hala nola leherketa bat edo bloke baten desagerpena.

```
1    public void setMota(int motaBerria) {
2        this.mota = motaBerria;
3        setChanged();
4        notifyObservers();
5
6    }
```

1.8 Jokalaria klasea

1.8.1 Atributuak

- **private boolean bizitza:** "Boolean" motako atributu pribatu bat da, jokalariak bizirik jarraitzen badu irudikatzen duena. True gisa hasten da, jokalaria jokoan aktibo dagoela adieraziz. Jokalariak bizitza galtzen badu, balio hori aldatu egingo da, eta false izatera pasako da.

- **private boolean bonbaPrest:** "Boolean" motako atributu pribatu bat da, jokalaria bonba bat jartzeko prest dagoen adierazten duena. True gisa hasten da, eta horrek esan nahi du jokalariak bonbak jar ditzakeela hasieran. Bonben kopurua (*bonbaKop*) zerora iristen bada, balio hori faltsutu egiten da, eta gehiago jartzea eragozten du.
- **private int bonbaKop:** "Int" motako atributu pribatu bat da, jokalariak jar ditzakeen bonben kopurua adierazten duena. 10. balioarekin hasten da. Bonba bat jartzen duen bakoitzean, murriztu egiten da kontagailua.

1.8.2 Eraikitzailea

Jokalaria(boolean pBizitza, boolean pBonbaPrest, int pBonbaKop) klasearen eraikitzailea da eta hiru parametro jasotzen ditu: *pBizitza*, *pBonbaPrest* eta *pBonbaKop*. Hala ere, ez ditu erabiltzen eta, horren ordez, balio finkoak esleitzen dizkie atributuei (true, true, 10). Horrek bermatzen du jokalaria beti bizitza aktibo batekin eta hamar bonbekin hastea.

```
1    public Jokalaria (boolean pBizitza, boolean pBonbaPrest, int pBonbaKop) {
2                        this.bizitza = true;
3                        this.bonbaPrest = true;
4                        this.bonbaKop = 10;
5          }
```

1.8.3 bonbaJarri() metodoa

Metodo publiko bat da, jokalariari bonba bat jartzen uzten diona baldin eta *bonbaPrest* true bada eta *bonbaKop* zero baino handiagoa bada. BonbaKop 1 murrizten du bonba jarri ondoren. *BonbaKop* zerora iristen bada, *bonbaPrest* falsean ezartzen da, jokalariak bonba gehiago jartzea saihestuz berriak lortu arte.

```
1    public void BonbaJarri () {
2            if (bonbaPrest && bonbaKop > 0) {
3              //Bonba bota
4              bonbaKop -= 1;
5
6              if (bonbaKop == 0) {
7                bonbaPrest = false;
8              }
9            }
10         }
```

1.8.4 bonbaBerria() metodoa

Eskuragarri dauden bonben kopurua batean handitzen duen metodo da (*bonbaKop*). Gainera, *bonbaPrest* faltsutzen bada, true bihurtzen da, eta jokalariari

berriz ere bonbak jartzeko aukera ematen dio. "If (true)" baldintza erredundantea bada ere, metodoa erabilgarria da bonbaz berriro hornitzeko partidan zehar.

```
1     public void BonbaBerria () {
2             if (true) {
3               bonbaKop += 1;
4
5               if (!bonbaPrest) {
6                 bonbaPrest = true;
7               }
8             }
9           }
```

1.9 JokalariEredua klasea

1.9.1 Atributuak

- **private int X:** Jokalariak X. ardatzean duen egungo posizioa adierazten du. Jokalaria mugitzen denean eguneratzen den balio oso bat da. Jolasaren barruan talkak edo desplazamenduak kalkulatzeko erabiltzen da, pertsonaiak mapan duen kokapen horizontala definitzen baitu.
- **private int Y:** Jokalariak Y. ardatzean duen egungo posizioa adierazten du. Zenbakizko balio oso bat da, eta jokoaren munduan duen kokapen bertikala adierazten du. Mugimendu bakoitzarekin eguneratzen da, eta beste objektu edo jokalari batzuekiko interakzioak zehazteko aukera ematen du.
- **private int aurreX:** Gorde jokalariaren aurreko posizioa X. ardatzean. Balio hori baliagarria da mugimenduak iraultzeko edo desplazamenduarekin lotutako kalkuluak egiteko. Jokalariak atzera egin duen edo norabidez aldatu den zehazteko aukera ematen du.
- **private int aurreY:** Jokalariaren aurreko posizioa Y ardatzean gordetzen du. Uneko posizioa aurrekoarekin alderatzeko balio du, eta horrek mekanikak inplementatzea errazten du, hala nola mugimenduak desegitea edo pertsonaian egoera-aldaketak detektatzea.
- **private boolean bizitza:** Jokalariak jokoaren barruan bizirik jarraitzen duen edo hil den adierazten du. Balio true batek esan nahi du pertsonaiak oraindik jolaz dezakeela partida, eta false izateak, berriz, bizitza galdu duela adieraziko luke.
- **private int bonbaKop = 10:** Jokalariak zenbat bonba jar ditzakeen adierazten du. Lehenetsitako 10ekin hasten da, eta murriztu egiten da bonba bat erabiltzen den bakoitzean. Haren balioa ezin da negatiboa izan, eta 0ra iristen denean, tenporizadorea aktibatzen da.

- **private boolean timerAktibatu = false:** Tenporizadorea aktibo dagoen adierazten du. Bere helburua bonben birkarga kontrolatzea da, eta jokalaria bonba horiek gabe geratzen denean aktibatzen da. Huts egiten duen bitartean, tenporizadoreak ez du ekintzarik egiten.
- **private boolean bonbaPrest = false:** Adierazten du ea bonba berri bat erabiltzeko prest dagoen tenporizadorea igaro ondoren. True bihurtzen da itxaronaldia amaitzen denean, eta jokalariari berriz ere bonbak jartzeko aukera ematen dio.
- **private int kont:** Bonba bat berriro erabili aurretik geratzen den denbora gordetzen duen kontrol-aldagaia da. *PERIODO* balioarekin hasten da, eta pixkanaka gutxitzen da 0ra iritsi arte; une horretan, bonba prest egongo da.
- **private int PERIODO = 6** Bonba berri bat prest egon aurretik igaro behar duen denbora-kantitatea definitzen du, segundotan. Haren balioa konstantea da klasearen barruan, eta kargatzeko denbora beti berdina izango dela bermatzen du.
- **private Timer timer = null:** Tenporizadorearen exekuzioa kudeatzen duen objektua. Aldian-aldian exekutatu behar diren atazak programatzeko erabiltzen da, hala nola kont murriztea. BonbaKop 0ra iristen denean aktibatzen da.

1.9.2 Eraikitzailea

JokalariEredu(int pX, int pY): Jokalariaren posizioa abiarazten duen eraikitzailea (pX, pY). Halaber, $aurreX$ eta $aurreY$ balioak esleitzen dizkio hasierako posizioari, eta jokalaria bizirik dagoela ezartzen du ($bizitza = true$).

```
1     public JokalariEredu(int pX, int pY) {
2                 this.X = pX;
3                 this.Y = pY;
4                 this.aurreX = pX;
5                 this.aurreY = pY;
6         this.bizitza = true;
7         }
```

1.9.3 getX() metodoa

X atributuaren uneko balioa itzultzen duen metodo publikoa da, hau da, jokalariak mapan duen posizio horizontala. Jolasaren beste zati batzuetan erabiltzen da pertsonaia non dagoen zehazteko eta maila barruan objektuekin, etsaiekin edo oztopoekin izan daitezkeen elkarreraginak kalkulatzeko.

```
1     public int getX() {
2                 return this.X;
3         }
```

1.9.4 getY() metodoa

Y atributuaren balio eguneratua itzultzen duen metodo publikoa, hau da, jokalariak jokoan duen posizio bertikala. *GetX()* -en antzeko erabilera du, eta funtsezko informazioa ematen du interfaze grafikoa eguneratzeko eta inguruko elementuekiko talkak edo interakzioak detektatzeko.

```
1     public int getY() {
2                  return this.Y;
3         }
```

1.9.5 setX(int i) metodoa

Metodo publiko bat da, X balioa parametro gisa jasotako i osoarekin eguneratzen duena. Jokalaria ardatz horizontalean mugitzeko erabiltzen da, posizio berria erregistratuz. Metodo hori funtsezkoa da mugimendu-mekanikarako eta pertsonaia jolasean mugitzeko logikarako.

```
1     public void setX(int i) {
2
3                  this.X = i;
4         }
```

1.9.6 setY(int i) metodoa

Y atributuari i balioa esleitzen dion metodo publikoa da, jokalariaren posizio bertikala aldatuz. Mugimendu-mekanikan erabiltzen da mapan duen kokapena aldatzeko, eta jolasaren logikak pertsonaiaren egungo posizioa interfaze grafikoan behar bezala islatzea ahalbidetzen du.

```
1     public void setY(int i) {
2
3                  this.Y = i;
4         }
```

1.9.7 getAurrekoX() metodoa

AurreX atributuaren balioa itzultzen duen metodo publikoa da, hau da, jokalariaren aurreko posizio horizontala azken mugimenduaren aurretik. Baliagarria da pertsonaiak jolasaren barruan izan duen ibilbidea kalkulatzen laguntzeko.

```
1     public int getAurrekoX() {
2                  return this.aurreX;
3         }
```

1.9.8 getAurrekoY() metodoa

AurreY-ren balioa itzultzen duen metodo publikoa, jokalariaren aurretiko posizio bertikala biltegiratuz. Aurreko mugimendua egiaztatzeko eta, behar izanez gero, doikuntzak egiteko balio du, eta mekanikak inplementatzeko aukera ematen du, hala nola talkak edo, errorerik izanez gero, mugimenduak desegiteko.

```
1     public int getAurrekoY() {
2                  return this.aurreY;
3         }
```

1.9.9 setAurreko(int i) metodoa

Metodo publiko horrek *i* balioa esleitzen dio *aurreX* atributuari, eta aurreko posizio berri bat gordetzen du X ardatzean, jokalaria mugitu aurretik. Baliagarria da posizioen historiala mantentzeko eta, beharrezkoa bada, jokoaren logikaren barruan aurretiko kokapenak berrezartzeko.

```
1     public void setAurrekoX(int i) {
2
3                  this.aurreX = i;
4         }
```

1.9.10 setAurrekoY(int i) metodoa

Metodo publiko horrek *i* balioa esleitzen dio *aurreY* atributuari, eta mugimendua egin aurreko azken posizio bertikala eguneratzen du. Jokalariak ingurunearekin dituen elkarreraginak maneiatzeko eta posizioen arteko trantsizio arina bermatzeko erabiltzen da, kokapen-akatsik gabe.

```
1     public void setAurrekoY(int i) {
2
3                  this.aurreY = i;
4         }
```

1.9.11 setBizitza(boolean b) metodoa

Metodo publiko bat da, *bizitza* atributuari *b* balioa esleitzen diona, jokalariaren bizi-egoera aldatuz. Falsean ezartzen bada, pertsonaia ezabatu egin dela edo bizia galdu duela adieraziko du. Funtsezko metodoa da jokalariaren bizirauenaren mekanika kudeatzeko.

```
1     public void setBizitza(boolean b) {
2                  bizitza=b;
3
4         }
```

1.9.12 bonbaNahiko() metodoa

Jokalariak bonba bat gutxienez erabilgarri badu, true itzultzen da (bonbaKop! = 0). Bestela, falsea itzultzen du, bonba gehiago ezin direla jarri adieraziz.

```
    public boolean bonbaNahiko() {
                return bonbaKop!=0;
        }
```

1.9.13 bonbaGutxitu() metodoa

Bonba gutxiagotzen duen metodoa da (*bonbaKop*–). Bonbak 0ra iristen badira, tenporizadorea aktibatzen da *timerAktibatu()* deituz.

```
    public void bonbaGutxitu() {
                this.bonbaKop--;
                if (bonbaKop==0)
                        timerAktibatu();
        }
```

1.9.14 timerAktibatu() metodoa

Tenporizadore bat aktibatzen du, jada gauzatzen ari ez bada. Segundu bakoitzean *kont* gutxitzen den ataza bat hasten du 0ra iritsi arte, eta une horretan *bonbaPrest* = *true* markatzen da.

```
    public void timerAktibatu() {
                if (!timerAktibatu) {
                        kont = PERIODO;
                        TimerTask timerTask = new TimerTask() {
                                @Override
                                public void run() {
                                        updateKont();
                                }
                        };
                        timer = new Timer();
                        timer.scheduleAtFixedRate(timerTask, 0, 1000);
                }
        }
```

1.9.15 denboraPasa() metodoa

True itzultzen du bonba bat jartzeko prest badago (*bonbaPrest* = *true*). Jokalariak itxaron ondoren bonbak berriro erabil ditzakeen zehazteko balio du.

```
    public boolean denboraPasa() {
                return bonbaPrest;
        }
```

1.9.16 updateKont() metodoa

Metodo pribatu bat da, *kont* segundoko 1 murrizten duena. *Kont* 0ra iristen denean, *PERIODO*ra berrabiatzen da, *bonbaPrest* = *true* aktibatzen da eta tenporizadorea baliogabetzen da.

```
1     private void updateKont() {
2                 kont--;
3                 if (kont == 0) {
4                         kont = PERIODO;
5                         bonbaPrest = true;
6                         System.out.print("\nBonba␣prest␣dago");
7                         timer.cancel();
8                 }
9         }
```

1.9.17 bonbaJarrita() metodoa

Jokalariak bonba bat jartzen duenean exekutatzen da. Aldatzen du *bonbaPrest*-a false eta desaktibatzen du tenporizadorea (*timerAktibatu* = *false*), beste bonba bat jarri aurretik itxaron behar dela adieraziz.

```
1     public void bonbaJarrita() {
2                 bonbaPrest = false;
3                 timerAktibatu = false;
4         }
```

1.10 LaberintoEredua klasea

1.10.1 Atributuak

- **private static LaberintoEredua nLE = null:** *LaberintoEredua* klasearen beraren instantzia estatikoa da. *Singleton* patroia inplementatzeko erabiltzen da, joko osoan labirintoaren instantzia bakarra dagoela ziurtatuz. Null-en hasten da eta *getLabEredua()* metodoaren bidez esleitzen da, beharrezkoa denean.
- `private ArrayList<GelaxkaEredu> gelaxkaZerrenda = null` Labirintoaren gelaxkak gordetzen dituen zerrenda bat da (`ArrayList<GelaxkaEredu>`). Gelaxka bakoitzak elementu mota bat adierazten du jolasaren barruan (bloke gogorrak, bloke bigunak, espazio hutsak, etab.). Null-en hasten da eta lehen aldiz sartzen denean eskatzen da.
- **private BlokeZerrenda bZerr:** BlokeZerrenda klaseko instantzia bat da, labirintoaren barruko blokeen bilduma kudeatzen duena. Jokoaren bloke gogorrak eta bigunak biltegiratzen eta maneiatzen ditu, eta objektu horiek partidan manipulatu eta ezabatzeko aukera ematen du.

- **private final int zutabe = 17:** Balio konstante bat da (final int), labirintoaren laukian guztira zenbat zutabe (17) dauden adierazten duena. Matrizearen mugak ezartzeko eta jokoan posizioak kalkulatzeko erabiltzen da.
- **private final int errenkada = 11:** Balio konstante bat da (final int), labirintoaren laukian dagoen errenkada kopuru osoa (11) definitzen duena. Gelaxken banaketa egituratzeko eta maparen barruan koordenatuak kalkulatzeko balio du.
- **private JokalariEredu bonberman:** *JokalariEredu*ko instantzia bat da, jokoaren pertsonaia nagusia irudikatzen duena. (0,0) posizioan hasten da, eta labirintoaren barruan duen hasierako kokapena zehazten du.
- **private boolean borratu = false:** Labirintoaren barruan elementuren bat ezabatu behar den adierazten duen atributu boolearra da. Jokoaren egoera eguneratzeko eta objektuak ezabatzeko hainbat eragiketetan erabiltzen da.
- `private Queue<Bonba> bonbaLista = new LinkedList<Integer>();` *LinkedList<Bonba>* sisteman oinarritutako ilara bat da (*Queue<Integer>*), jokoaren simulazioan balio osoak kudeatzeko erabiltzen dena. Bonbak lehertzeko mekanikan eta kaltetutako blokeak ezabatzeko erabiltzen da.
- `private Queue<Sua> suLista = new LinkedList<Integer>();` Aurreko atributoan bezala, ilara bat da, zehatzago su ilara bat. Laberitoak bonba leher ostean sua jartzeko erabiltzen da. Izan ere, suak min egiten jarraitu behar du leherketaren ostean 2 segunduz.

1.10.2 Eraikitzailea

LaberintoEredua() klaseko eraikitzaile pribatua da. Blokeen zerrenda (*bZerr*) abiarazten du, labirintoaren matrizea sortzen du *matrizeaSortu()*-ren bidez eta jokalariarentzako *JokalariEredu* instantzia sortzen du. Halaber, labirintoaren egoera aldatu egin dela jakinarazten die behatzaileei.

```
1     private LaberintoEredua() {
2                 this.bZerr = new BlokeZerrenda();
3                 matrizeaSortu();
4                 this.bonberman = new JokalariEredu(0,0);
5                 //bZerr.;
6
7         }
```

1.10.3 getLabEredua() metodoa

Singleton eredua ezartzen duen metodo estatikoa da. *nLE* null bada, sortzen du *LaberintoEredua* objektu berri bat; berriz, jada sortuta badago jarraian itzultzen du. Exekuzioan labirintoaren irudikapen bakarra egongo dela ziurtatzen du.

```
    public static LaberintoEredua getLabEredua() {
                if (nLE == null) {
                        nLE = new LaberintoEredua();
                }
                return nLE;
        }
```

1.10.4 eguneratu() metodoa

eguneratu() metodoaren bitartez, Observers-ei laberintoan aldatu dena tranmititzen die, hau da, observer gisa jokatzen du. Oso metodo baliogarria da, interfaze grafikoan aldaketak emateko eta jokalaria mugitzeko.

```
    public void eguneratu() {
                // TODO Auto-generated method stub
                setChanged();
                notifyObservers(new int[] {1});
        }
```

1.10.5 matrizeaSortu() metodoa

Labirintoaren egitura sortzen duen metodo pribatua da, *gelaxkaZerrenda* gelaxka mota desberdinekin betez: bloke gogorrak, bloke bigunak, espazio hutsak eta Bonbermanen hasierako posizioa erabiltzen du matrizea sortzeko. *AusazZenbakia()* metodoa erabiltzen du elementuak ausaz jartzeko.

```
    private void matrizeaSortu() { //Hurrengo aukerak: 0 Gelaxka Hutsa, 1 Bloke Gogorra, 2
        Bloke Biguna 5 Bonberman-a
                int lerro, zut;
                for(lerro=0;lerro<errenkada;lerro++) {
                        for(zut=0;zut<zutabe;zut++) {
                                GelaxkaEredu gel = null;
                                int i = zut%17;
                                if (i%2!=0 && lerro%2!=0) {
                                        bZerr.gehituBloke(new Bloke("Gogorra", zut, lerro)
                                            );
                                        gel = new GelaxkaEredu(1);
                                        getGelaZerr().add(gel); //Bloke Gogorra
                                }
                                else if (ausazZenbakia()>0.4 && !((lerro==0 && zut==0)||(
                                    lerro==0 && zut==1)||(lerro==1 && zut==0))){
                                        bZerr.gehituBloke(new Bloke("Biguna", zut, lerro))
                                            ;
                                        gel = new GelaxkaEredu(2);
                                        getGelaZerr().add(gel); //Bloke Biguna
                                }
                                else if (lerro==0 && zut==0) {
                                        gel = new GelaxkaEredu(5);
```

```
19                                      getGelaZerr().add(gel); //Bonberman
20                                      bZerr.gehituBloke(new Bloke(null));
21                              }
22                              else {
23                                      gel = new GelaxkaEredu(0);
24                                      getGelaZerr().add(gel); //Hutsik
25                                      bZerr.gehituBloke(new Bloke(null));
26                              }
27                      }
28              }
29      }
```

1.10.6 getGelaZerr() metodoa

GelaxkaZerrenda gelaxken zerrenda itzultzen du. Oraindik ez bada hasi, sortu egiten du. Labirintoaren matrizea jolasaren beste leku batzuetan sartzeko eta kudeatzeko erabiltzen da.

```
1   public ArrayList<GelaxkaEredu> getGelaZerr() {
2               if(gelaxkaZerrenda == null) {
3                       gelaxkaZerrenda = new ArrayList<>();
4               }
5               return gelaxkaZerrenda;
6       }
```

1.10.7 getbZerr() metodoa

Labirintoko blokeak dituen *BlokeZerrenda*ren instantzia itzultzen du. Partidan dauden blokeetara sartzeko eta horiek aldatzeko aukera ematen du.

```
1   public BlokeZerrenda getbZerr() {
2               return bZerr;
3       }
```

1.10.8 getBomberman() metodoa

*JokalariEredu*k Bonberman bueltatzen du jokoan. Bere posizioan sartzeko eta bere egoera manipulatzeko erabiltzen da.

```
1   public JokalariEredu getBomberman() {
2               return this.bonberman;
3       }
```

1.10.9 getZutabe() metodoa

Zutabe kopuru osoa (*zutabe*) itzultzen du. Posizio-kalkuluak egiteko eta labirintoaren saretan mugak egiaztatzeko erabiltzen da.

```
    public int getZutabe() {
                return zutabe;
        }
```

1.10.10 getErrenkada() metodoa

Errenkada kopuru osoa itzultzen da (*errenkada*). Baliagarria da mapa egituratzeko eta matrizearen barruan posizioak zehazteko.

```
    public int getErrenkada() {
                return errenkada;
        }
```

1.10.11 getBorratu() metodoa

Ezabatu balio boolearra (*borratu*) itzultzen du. Labirintoko elementuren bat ezabatu edo aldatu behar den egiaztatzeko aukera ematen du.

```
    public boolean getBorratu() {
                return this.borratu;
        }
```

1.10.12 setBorratu(boolean borratu) metodoa

borrar atributuari balio berri bat esleitzen dio, eta jokoan elementuak ezabatu behar diren aldatzen du.

```
    public void setBorratu(boolean borratu) {
                this.borratu = borratu;
        }
```

1.10.13 ausazZenbakia() metodoa

Ausazko zenbaki bat (*double*) 0 eta 1 artean sortzen duen metodo pribatua, *Random.nextDouble()* erabiliz. Labirintoan ausaz bloke bigunak eta etsaiak agertzea zehazteko erabiltzen da.

```
    private double ausazZenbakia() {
                Random ram = new Random();
                double aukera = ram.nextDouble();
                return aukera;
        }
```

1.10.14 mugitu(int i, int j) metodoa

Bonberman norabidean (i, j), posizio berria baliozkoa den mugitzen du eta bloke gogor edo bigun edo bonba batekin talka egiten ez duela egiaztatuz. Mugimendua posible bada, eguneratuzen du pertsonaiaren posizioa matrizean, eta doitzen dugelaxka mota aldaketa islatzeko.

```
    public void mugitu(int i, int j) \{

boolean mugitu = false;

boolean erre = false;

boolean bonba = false;

if (!((this.bonberman.getX()+i<0) || (this.bonberman.getY()+j<0) || (this.bonberman.getX()
    +i>16) || (this.bonberman.getY()+j>10)))\{ //Bonberman-a tarteetan dagoela konprobatzen
     du

if(!((getGelaZerr().get((this.bonberman.getY()+j)*17+this.bonberman.getX()+i).getMota() ==
     1) || (getGelaZerr().get((this.bonberman.getY()+j)*17+this.bonberman.getX()+i).getMota
    () == 2) || (getGelaZerr().get((this.bonberman.getY()+j)*17+this.bonberman.getX()+i).
    getMota() == 3) || (getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX())
    .getMota() == 14))){ //Bonberman-a blokeekin ez dela joko konprobatzen du

this.bonberman.setAurrekoX(\textbf{this}.bonberman.getX()); //Lehengo, aurreko posizioak
    eguneratzen ditu eta gero helduko den posizioa


this.bonberman.setAurrekoY(\textbf{this}.bonberman.getY());

if (getGelaZerr().get((this.bonberman.getY()+j)*17+this.bonberman.getX()+i).getMota() ==
    4) {

for(Sua su : suLista) {

if (su.getX() == (this.bonberman.getX()+i) && su.getY() == (this.bonberman.getY()+j)) {

su.ezkonduSua();

}

}

getGelaZerr().get((this.bonberman.getY()+j)*17+this.bonberman.getX()+i).setMota(14);

erre = true;

}

```

```
36 if (getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).
       getMota() == 15) {
37
38 getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).setMota
       (3);
39
40 bonba = true;
41
42 }
43
44 this.bonberman.setX(this.bonberman.getX()+i);
45
46 this.bonberman.setY(this.bonberman.getY()+j);
47
48 mugitu = true;
49
50 }
51
52 }
53
54 if(!erre && !(getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).getMota
       () == 15) && !(getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).
       getMota() == 14)) \{
55
56 switch (i)\{ //Mugimenduaren arabera "sprite" bat edo beste bat erakutsiko du
57
58 case -1:
59
60 if(getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).
       getMota() == 6 || getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).
       getMota() == 6){
61
62 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(7);
63
64 }
65
66 else {
67
68 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(6);
69
70 }
71
72 break;
73
74 case 1:
75
76 if(getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).
       getMota() == 8 || getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).
       getMota() == 8){
77
```

```
78 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(9);
79
80 }
81
82 else {
83
84 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(8);
85
86 }
87
88 break;
89
90 }
91
92 switch (j){
93
94 case -1:
95
96 if(getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).
       getMota() == 12 || getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).
       getMota() == 12){
97
98 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(13);
99
100 }
101
102 else {
103
104 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(12);
105
106 }
107
108 break;
109
110 case 1:
111
112 if(getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).
       getMota() == 10 || getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).
       getMota() == 10){
113
114 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(11);
115
116 }
117
118 else {
119 getGelaZerr().get((this.bonberman.getY())*17+this.bonberman.getX()).setMota(10);
120
121 }
122
123 break;
```

```
124
125 }
126
127 }
128
129 if (mugitu \&& !bonba) //Gelaxka hutsik dagoela adierazteko balio du, alegia, Bonberman-a
    mugitu da, baldin eta bonbarik jarri ez badu.
130
131 getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.bonberman.getAurrekoX()).setMota
    (0);
132
133 if(erre) {
134
135 System.out.print("\textbackslash{}nPartida␣galdu␣duzu.");
136
137 javax.swing.Timer timer = new javax.swing.Timer(5, e -> partidaBukatu());
138
139 timer.setRepeats(false);
140
141 timer.start(); // Timer-a martxan jarri
142
143 }
144
145 }
```

1.10.15 bonbaJarri() metodoa

Bonba bat Bonbermanen egungo posizioan jartzen du, eta gorde proba-ilaran eta dagoen gelaxka bonba batean bihurtzen du, dagoen bonba islatzeko. Bonba hori segundu batzuk pasata lehertuko da.

```
1     public void bonbaJarri() {
2
3 Bonba bonba;
4
5 if (bonberman.bonbaNahiko()) {
6
7 bonberman.bonbaGutxitu();
8
9 bonba = new Bonba(3, this.bonberman.getX(), this.bonberman.getY());
10
11 getGelaZerr().get(this.bonberman.getY()*17+this.bonberman.getX()).setMota(15);
12
13 bonbaLista.add(bonba);
14
15 }
16
17 else if} (bonberman.denboraPasa()){
18
```

```
bonba = new Bonba(3, this.bonberman.getX(), this.bonberman.getY());

getGelaZerr().get(this.bonberman.getY()*17+this.bonberman.getX()).setMota(15);

bonberman.bonbaJarrita();


bonberman.timerAktibatu();

bonbaLista.add(bonba);

}
}
```

1.10.16 apurtuBlokeak (int zut, int lerr) metodoa

Posizioaren inguruko blokeak suntsitzen ditu (*zut*, *lerr*), bloke bigunak diren egiaztatuz eta gelaxka sugar bihurtuz (*Sua*). Bonberman leherketa-eremuan badago, haren egoera eguneratuko da, harrapatu dutela, bizitza false bihurtuz eta jokotik kendu dutela adieraziz.

```
    public void apurtuBlokeak(int zut, int lerr) {
                int pos = lerr*17+zut;
                bZerr.blokeakInprimatu();
                proba.remove();
                System.out.print(proba.peek());
                if(zut-1>=0 && !(getGelaZerr().get(lerr*17+(zut-1)).getTipo() == 1)) {
                        pos = pos-1;
                        Sua sua = new Sua(zut-1,lerr);
                        if (this.bonberman.getY()*17+this.bonberman.getX() == lerr*17+(zut
                            -1)) {
                                getGelaZerr().get(lerr*17+(zut-1)).setTipo(12);
                                this.bonberman.setBizitza(false);
                        }
                        else
                                getGelaZerr().get(lerr*17+(zut-1)).setTipo(4);
                        bZerr.getBloke(lerr*17+(zut-1));
                        System.out.print("\n"+pos);
                }
                if(zut+1<=16 && !(getGelaZerr().get(lerr*17+(zut+1)).getTipo() == 1)) {
                        pos = pos+1;
                        Sua sua = new Sua(zut+1,lerr);
                        if (this.bonberman.getY()*17+this.bonberman.getX() == lerr*17+(zut
                            +1)) {
                                getGelaZerr().get(lerr*17+(zut+1)).setTipo(12);
                                this.bonberman.setBizitza(false);
                        }
                        else
                                getGelaZerr().get(lerr*17+(zut+1)).setTipo(4);
```

```
                        bZerr.getBloke(lerr*17+(zut+1));
                        System.out.print("\n"+pos);
                }
                if(lerr-1>=0 && !(getGelaZerr().get((lerr-1)*17+zut).getTipo() == 1)) {
                        pos = pos-17;
                        Sua sua = new Sua(zut,lerr-1);
                        if (this.bonberman.getY()*17+this.bonberman.getX() == (lerr-1)*17+
                            zut){
                                getGelaZerr().get((lerr-1)*17+zut).setTipo(12);
                                this.bonberman.setBizitza(false);
                        }
                        else
                                getGelaZerr().get((lerr-1)*17+zut).setTipo(4);
                        bZerr.getBloke((lerr-1)*17+zut).setMota(null);
                        System.out.print("\n"+pos);
                }
                if(lerr+1<=10 && !(getGelaZerr().get((lerr+1)*17+zut).getTipo() == 1)) {
                        pos = pos+17;
                        Sua sua = new Sua(zut,lerr+1);
                        if (this.bonberman.getY()*17+this.bonberman.getX() == (lerr+1)*17+
                            zut) {
                                getGelaZerr().get((lerr+1)*17+zut).setTipo(12);
                                this.bonberman.setBizitza(false);
                        }
                        else
                                getGelaZerr().get((lerr+1)*17+zut).setTipo(4);
                        bZerr.getBloke((lerr+1)*17+zut).setMota(null);
                        System.out.print("\n"+pos);
                }
        }
```

1.10.17 private void partidaBukatu()

partidaBukatu() metodoaren bitartez, partida amaitzeko aukera ematen zaigu, metodo honen bidez, jokalaria hiltzean partida amaitzen da.

```
private void partidaBukatu(){
denboraItxaron(100);
System.exit(0);
}
```

1.10.18 denboraItxaron()

Metodo honen bitartez, jokoa momentu batez gelditu daiteke amaitu aurretik.

```
private static void denboraItxaron(int pMillis) {
                 try { Thread.sleep(pMillis);
                 }
                 catch (Exception e) {}
```

```
5                   }
```

1.11 Sua klasea

1.11.1 Atributuak

- **private int X:** Laberintoan sua sortzen den posizioaren X koordenatua irudikatzen du. Eraikitzailean hasten da, pasatutako balioa parametro gisa hartuta, eta jokoaren matrizean duen kokapena zehazten du. Funtsezkoa da denbora-tarte baten ondoren sua non agertuko eta desagertuko den jakiteko.
- **private int Y:** Labirintoan sua sortzen den Y koordenatua irudikatzen du. Haren balioa eraikitzailean esleitzen da, eta kalkuluetan labirintoaren gelaxkaren egoera aldatzeko erabiltzen da. Funtsezkoa da sua aktibatzean haren egoera zer posiziotan aldatuko den zehazteko.
- **private int kont:** Sua desagertu aurretik zenbat denboraz egongo den ikusgai adierazten duen kontagailu bat da. PERIODO balioarekin hasten da, eta murriztu egiten da tenporizadorearen iterazio bakoitzean. Zerora iristen denean, sua ezabatu egiten da eta kontagailua berrabiarazten da etorkizuneko exekuzioetarako.
- **private int PERIODO = 2:** Sua desagertu aurretik gelaxkan egoten den denbora segundotan definitzen du. *Kont* kontagailua hasieratzeko erabiltzen da, eta erreferentzia gisa balio du tenporizadorearen logikan. Bere balio lehenetsia 2 da, eta horrek esan nahi du suak desagertu aurretik bi segundo irauten duela.
- **private Timer timer = null:** *Timer* klaseko objektu bat da, eta bigarren planoan zereginak aldizka gauzatzea kudeatzen du. Laberintoaren gelaxkako suaren iraupena kontrolatzeko erabiltzen da. Eraikitzailean hasten da eta *updateKont()* metodoa exekutatzen du segundo bakoitzean, denbora bete arte.
- **private boolean piztuta = false:** Atributu hau boolean motatakoa da, eta bere funtzio nagusia sua piztuta edo itzalita dagen zehaztea da. Oso baliogarria da sua bi segundu pasata desagerrarazteko.

1.11.2 Eraikitzailea

Sua(int pX, int pY): *Sua*ko instantzia bat koordenatuetan (pX, pY) abiarazten duen eraikitzailea da, X, Y eta *kont* koordenatuei balioak esleituz. TimerTask bat ere sortzen du, segundo bakoitzean *updateKont()* metodoa exekutatzen duena, suaren iraupena kontrolatuz. Denbora amaitutakoan, gelditu tenporizadorea eta ezabatu sua.

```
1       public Sua(int pX, int pY){
2           this.X = pX;
```

```
        this.Y = pY;
        kont = PERIODO;
        TimerTask timerTask = new TimerTask() {
                @Override
                public void run() {
                        updateKont();
                }
        };
        timer = new Timer();
        timer.scheduleAtFixedRate(timerTask, 0, 1000);
        }
```

1.11.3 updateKont() metodoa

Segundoro exekutatzen da *Timer*-ri esker. *Kont* kontagailua batean murrizten du, eta zerora iristen denean, *ezkonduSua()* metodoari deitzen dio sua laberintotik kentzeko. *PERIODOA*-ren balioari berriro ekin ondoren, *Timer*-aren exekuzioa amaitzen da *timer.cancel()* erabiliz.

```
    private void updateKont() {
                kont--;
                if (kont == 0) {
                        kont = PERIODO;
                        ezkonduSua();
                        timer.cancel();
                }

        }
```

1.11.4 ezkonduSua() metodoa

Sartu egiten da laberintoaren instantziara eta aldatu (X, Y) dagokion gelaxka mota, jada surik ez dagoela adierazteko. Jolasean sua bistatik desagerrarazteko erabiltzen da, gelaxka denbora jakin baten ondoren jatorrizko egoerara itzuliko dela ziurtatuz.

```
    public void ezkonduSua() {

LaberintoEredua.getLabEredua().getGelaZerr().get(Y * 17 + X).setMota(0);

}
```

1.11.5 getX() metodoa

Metodo honen bidez, suak momentuak duen posizio horizontala jakitea posible dugu, suaren "X" posizioa itzultzen baitdigu.

```
1         public int getX() {
2                  return this.X;
3         }
```

1.11.6 getY() metodoa

getY() metodoaren bitartez, suaren posizio bertikala jasotzen dugu, hau da zein matrizearen "Y" posizioan dagoen.

```
1     public int getY() {
2                  return this.Y;
3         }
```

2 Kontroladoreen ekintzak

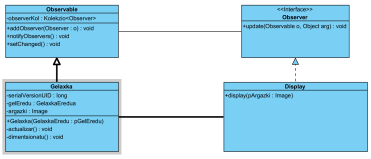
Kontroladore metodoa *LaberintoBista* klasekoa da, KeyListener interfazea inplementatzen du eta jokoan teklatu-sarrera maneiatzeaz arduratzen da. Bere helburu nagusia erabiltzaileak sakatutako teklak detektatzea eta jokoaren logikan dagozkion ekintzak gauzatzea da.

Erabiltzaileak tekla bat sakatzen duenean, *keyPressed* metodoa (*KeyEvent e*) automatikoki deitzen zaio eta *e.getKeyCode()* erabiliz zer tekla sakatu den ebaluatzen du. Teklaren kodearen arabera, ekintza hauek egiten dira:

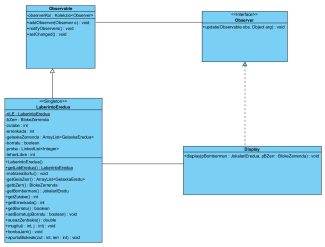
1. **Bomberman-en mugimendua:**
   (a) Eskuineko gezia sakatzen bada (*KeyEvent. VK_RIGHT*), *lE.mugitu metodoa (1,0)*; pertsonaia eskuinerantz mugitzen du lauki bat.
   (b) Tekla sakatuz gero, gezia gora (*KeyEvent. VK_UP*), *lE.mugitu(0, -1)*; gorantz mugitzen du.
   (c) Tekla sakatuz gero, gezia behera (*KeyEvent. VK_DOWN*), *lE.mugitu(0,1)*; beherantz mugitzen du.
   (d) Ezkerreko gezia sakatzen bada (*KeyEvent. VK_LEFT*), *lE.mugitu(-1,0)*; ezkerrerantz mugitzen du.
2. **Bonbak jartzea**
   (a) Erabiltzaileak A tekla sakatzen badu (*KeyEvent. VK_A*), *lE.bonbaJarri()* exekutatzen da; horren ondorioz, Bomberman-ek ponpa bat jartzen du egungo posizioan.

3 Observer Klase Diagrama

Hona hemen Gelaxka klaseko Observer KD:



Eta hemen LaberintoEredua-ko observer klase diagrama:



4 Notify/Update inplementazioa

Observer patroia proiektuaren hainbat motatan inplementatzen da, modeloaren datuetan aldaketak gertatzen direnean interfaze grafikoa eta jokoaren logika automatikoki eguneratzeko. Fluxu orokorra honako hau da:

1. **setChanged() eta notifyObservers():** *Observer* zabaltzen duten klaseetan erabiltzen dira, datuak aldatu zirela adierazteko eta behatzaileei jakinarazteko.
2. **update(Observable o, Object arg):** *Observer* zabaltzen duten klaseetan ezartzen da, aldaketen aurrean erreakzionatzeko eta ikusmen-egoera edo egoera logikoa eguneratzeko.

4.1 notifyObservers()

Atal honetan, *notifyObservers()* non erabiltzen diren azaltzen da, eta erabilitako metodo bakoitzean zer funtzio duen.

4.1.1 GelaxkaEredu

setTipo(int nuevoTipo) metodoan kokatuta dago eta bertan, gelaxka mota aldatzen den bakoitzean (*mota*), aldatutzat markatzen da (*setChanged()*) eta behatzaileak jakinarazten dira (*notifyObservers()*) eta horrek gelaxkari lotutako edozein elementu bisual edo logiko berehala eguneratzen dela ziurtatzen du.

```
1    public void setTipo(int nuevoTipo) {
2        this.mota = nuevoTipo;
3        setChanged();
4        notifyObservers();
5    }
```

4.1.2 LaberintoEredua

1. **Eraikitzailean** erabiltzen da non *LabirintoEredua* labirintoaren eredua da, eta labirintoaren egoera eta logika ditu.

```
1        private LaberintoEredua() {
```

```
                this.bZerr = new BlokeZerrenda();
                matrizeaSortu();
                this.bonberman = new JokalariEredu(0,0);

                setChanged();
                notifyObservers();
        }
```

2. **mugitu(int i, int j) metodoa:**Jokalaria mugitzen denean (*mugitu ()*), posizioak eguneratzen dira eta behatzaileei jakinarazten zaie.

```
        public void mugitu(int i, int j) {
                boolean mugitu = false;
                boolean erre = false;
                if (!((this.bonberman.getX()+i<0) || (this.bonberman.getY()+j<0) || (
                    this.bonberman.getX()+i>16) || (this.bonberman.getY()+j>10))){
                    //Bonberman-a tarteetan dagoela konprobatzen du
                        if(!((getGelaZerr().get((this.bonberman.getY()+j)*17+this.
                            bonberman.getX()+i).getTipo() == 1) || (getGelaZerr().get((
                            this.bonberman.getY()+j)*17+this.bonberman.getX()+i).
                            getTipo() == 2) || (getGelaZerr().get((this.bonberman.getY
                            ()+j)*17+this.bonberman.getX()+i).getTipo() == 3))){
                                    //Bonberman-a blokeekin ez dela joko konprobatzen
                                du
                                this.bonberman.setAurrekoY(this.bonberman.getY());
                                        //Lehengo aurreko posizioak eguneratzen ditu
                                    eta gero helduko den posizioa
                                this.bonberman.setY(this.bonberman.getY()+j);
                                this.bonberman.setAurrekoX(this.bonberman.getX());
                                this.bonberman.setX(this.bonberman.getX()+i);
                                mugitu = true;
                                if (getGelaZerr().get((this.bonberman.getY())*17+this.
                                    bonberman.getX()).getTipo() == 4) {
                                        getGelaZerr().get((this.bonberman.getY())*17+
                                            this.bonberman.getX()).setTipo(12);
                                        erre = true;
                                }
                                else {
                                        getGelaZerr().get((this.bonberman.getY())*17+
                                            this.bonberman.getX()).setTipo(5);
                                        System.out.print("Mugitu");
                                }
                        }
                }

                if(!erre) {
                        switch (i){     //Mugimenduaren arabera "sprite" bat edo beste
                             bat erakutsiko du
                        case 0:
                                getGelaZerr().get((this.bonberman.getY())*17+this.
```

```
                                    bonberman.getX()).setTipo(getGelaZerr().get((this.
                                    bonberman.getY())*17+this.bonberman.getX()).getTipo
                                    ());
26                                  break;
27                          case -1:
28                                  getGelaZerr().get((this.bonberman.getY())*17+this.
                                    bonberman.getX()).setTipo(6);
29                                  break;
30                          case 1:
31                                  getGelaZerr().get((this.bonberman.getY())*17+this.
                                    bonberman.getX()).setTipo(7);
32                                  break;
33                          }
34
35                          switch (j){
36                          case 0:
37                                  getGelaZerr().get((this.bonberman.getY())*17+this.
                                    bonberman.getX()).setTipo(getGelaZerr().get((this.
                                    bonberman.getY())*17+this.bonberman.getX()).getTipo
                                    ());
38                                  break;
39                          case -1:
40                                  if(getGelaZerr().get((this.bonberman.getAurrekoY())
                                    *17+this.bonberman.getAurrekoX()).getTipo() == 8){
41                                          getGelaZerr().get((this.bonberman.getY())*17+
                                            this.bonberman.getX()).setTipo(10);
42                                  }
43                                  else {
44                                          getGelaZerr().get((this.bonberman.getY())*17+
                                            this.bonberman.getX()).setTipo(8);
45                                  }
46                                  break;
47                          case 1:
48                                  getGelaZerr().get((this.bonberman.getY())*17+this.
                                    bonberman.getX()).setTipo(9);
49                                  break;
50                          }
51                  }
52
53                  if (mugitu) //Gelaxka hutsik dagoela adierazteko balio du, alegia,
                        Bonberman-a mugitu da.
54                          getGelaZerr().get((this.bonberman.getAurrekoY())*17+this.
                            bonberman.getAurrekoX()).setTipo(0);
55
56                  setChanged();
57                  notifyObservers();
58          }
```

4.2 update()

Klase horiek Observer zabaltzen dute, eta beha daitezkeen objektuen aldaketen aurrean erreakzionatzen dute, eta hurrengo motetan erabili dira.

4.2.1 Gelaxka klasea

update(Observable o, Object arg) metodoan erabili dugu eta bertan *GelaxkaEredu*ko behatzaile gisa harpidetzen da, eta, beraz, gelaxka mota aldatzen den bakoitzean, *update()* exekutatzen da, *update()* deia egiten duena, gelaxkari lotutako irudia berriz marraztuz.

```
1     public void update(Observable o, Object arg) {
2
3                  actualizar();
4         }
```

4.2.2 LaberintoBista klasea

update(Observable o, Object arg) metodoan erabili dugu eta *LaberintoEredu*ko behatzaile gisa harpidetzen da, eta horrek esan nahi du jokalaria mugitzen den bakoitzean edo labirintoaren egoera aldatzen den bakoitzean, interfaze grafikoa berriro sortzen dela *SortuLaberintoa()* -rekin.

```
1     public void update(Observable o, Object arg) {
2
3                  LaberintoEredua LE = LaberintoEredua.getLabEredua();
4                  sortuLaberintoa();
5
6
7         }
```

5 Atazen Dokumentazioa

Amaitzeko, jarraian atazen dokumentazioa nola egin den adierazteko taula:

Sprint	Ataza	Arduraduna	Denb. Planif.	Denb. Erreala	Oharrak
1	Diseinua	Mikel	2 Ordu	4 Ordu	
1	Programazioa	Asier Barrio	5 Ordu	7 Ordu	
1	Dokumentazioa	Gaizka	5 Ordu	4 Ordu	
1	Klaseen garapena	Asier LasHayas	4 Ordu	6 Ordu	