Innlevering 6

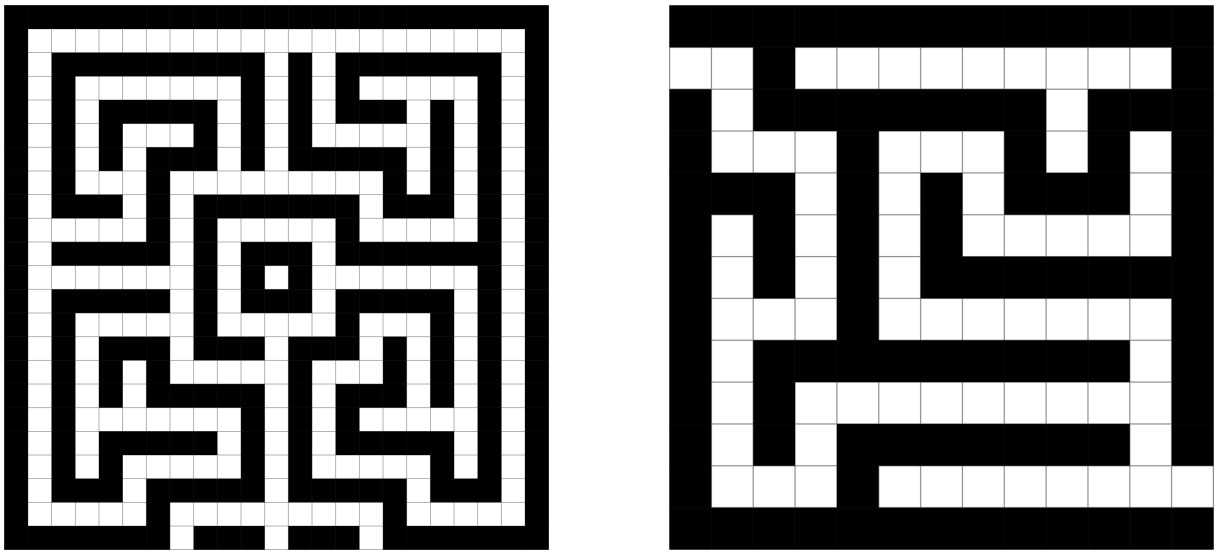
Innledning

I denne oppgaven skal du bruke rekursjon til å lage et program som kan finne veier ut av en labyrint. Labyrintene i denne oppgaven er rutenett, bygget opp av kvadratiske ruter som man enten kan gå gjennom eller ikke. Diverse labyrinter finnes lagret på fil, og filformatet blir beskrevet lenger nede i oppgaven.



Figur 1: Labyrint ved Egeskov slott i Danmark, designet av Piet Hein.

En labyrint er bygd opp av kvadratiske *ruter* som alle er like store. Vi kan referere til en bestemt rute ved å bruke koordinatene slik: **(radNr, kolonneNr)** der (0, 0) er øvre venstre hjørne. En rute kan være *hvit,* som betyr at man kan gå gjennom den, eller *sort,* som betyr at den utgjør en sperre. Vi sier at to ruter er *naboer* hvis de har en felles side. En rute kan altså ha inntil 4 naboer.



Figur 2: Til venstre en syklisk labyrint (fil `23x23.labyrint`) og til høyre en asyklisk labyrint (fil `13x13.labyrint`).

Vi kan gå fra én hvit rute til en annen hvit rute hvis de to er naboer. Vi kan sette sammen mange slike nabopar av hvite ruter for å danne en *vei*.

Vi sier at en vei er *syklisk* hvis en av rutene som ligger på veien, besøkes flere ganger; da er det fare for å gå i sirkel. Vi sier at en labyrint er syklisk dersom det finnes én eller flere sykliske veier i den; hvis ikke, kalles den *asyklisk*. Labyrinten til venstre i Figur 2 er syklisk fordi den inneholder en syklisk vei rundt midten; labyrinten til høyre er asyklisk.

Hvite ruter på kanten av labyrinten kalles *åpninger*; for eksempel har den høyre labyrinten i Figur 2 to åpninger: (1, 0) og (11, 12). Du kan anta at to åpninger aldri er naboer. En vei fra en gitt rute til en åpning kalles en *utvei*.

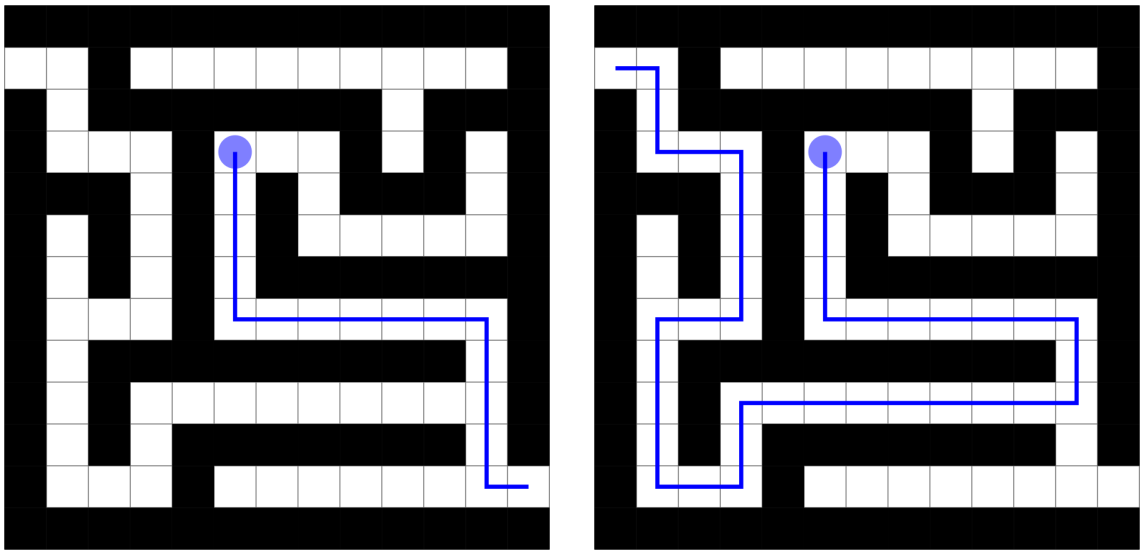
Merk: Vi kan ha områder med hvite ruter som er helt omringet av sorte ruter, for eksempel midten i den venstre labyrinten i Figur 2 eller området nordøst i den høyre. Dette innebærer at det ikke finnes noen utvei fra disse hvite rutene.

Filformat

For å representere labyrinter bruker vi følgende filformat:

```
7 9
#########
#........
#.#######
#.......#
#######.#
........#
#########
```

(Dette er fil `7x9.labyrint` i testfilmappen.)



Figur 3: De to utveiene fra (3, 5) i labyrinten i fil `13x13.labyrint` ender i åpningene (11,12) og (1,0).

- Den første linjen inneholder to positive heltall som bestemmer henholdsvis **antall rader** og **antall kolonner** i labyrinten.
- Hvite ruter representeres ved `.` (punktum).
- Sorte ruter representeres ved `#` (hashtag/firkant/skigard).
- Øvrige tegn (for eksempel blanke og linjeskift) ignoreres.

Relevante Trix-oppgaver

Alle oppgaver om rekursjon.

Oppgave 1 – Klasser og datastruktur

Under skisseres klassehierarkiet og datastrukturen som skal brukes. Du skal lage klassene `Labyrint`, `Rute`, `HvitRute`, `SortRute` og `Åpning`.

(a) `Labyrint`-klassen skal inneholde en todimensjonal array med referanser til alle `Rute`-objektene i labyrinten og den skal ta vare på antall rader og antall kolonner. Du skal redefinere metoden `toString` i `Labyrint`-klassen slik at labyrinten kan skrives ut med `System.out.println(lab)` der `lab` er en referanse til `Labyrint`-objektet.

Hint: I `toString` kan du legge inn linjeskift med `\n`.

(b) Klassen `Rute` skal ta vare på sine koordinater (radnummer og kolonnenummer) og ha en referanse til labyrinten den er en del av. I tillegg skal klassen ha referanser til sine eventuelle nabo-ruter (nord/syd/vest/øst). Det skal *ikke* være mulig å opprette et objekt av klassen `Rute`, kun av subklassene. Du bestemmer selv hva konstruktørene skal gjøre og hvilke parametre de skal ha.

(c) `HvitRute` og `SortRute` er subklasser av Rute. De skal redefinere metoden `toString`.

(d) `Åpning` er en subklasse av `HvitRute`.

Oppgave 2 – Innlesing fra fil

Programmet ditt skal ta et filnavn som parameter og gi det videre til konstruktøren i `Labyrint`-klassen. Konstruktøren leser filen og oppretter datastrukturen med den todimensjonale arrayen og alle `Rute`-objektene basert på dataene i filen. Programmet skal terminere om filen ikke finnes eller om filformatet er feil.

Hint: Hvis du bruker `Scanner` til lesingen og leser de to første heltallene i filen med `nextInt()`, må du etterpå kalle på `nextLine()` for å få lest det som står igjen på den første linjen i filen.

Hint: Før du begynner å lage `Rute`-objekter, bør du tenke over hvilken rekkefølge ting bør gjøres i og hvilke konsekvenser dette får for programmet ditt. For eksempel må du sørge for at alle ruter får kunnskap om sine naboer etter at `Rute`-objektene er laget.

Etter at konstruktøren har lest inn labyrinten fra fil, skal du teste at dette er gjort riktig ved et kall på `System.out.println(this)`.

Relevante Trix-oppgaver
01.19, *01.20* & *02.16*

Oppgave 3 – Løsning ved rekursjon

Du skal bruke rekursjon til å finne eventuelle åpninger det finnes en vei til fra en gitt startrute. Denne startrutens jobb blir å spørre alle sine naboruter om å finne en vei videre. Hvite naboruter spør så igjen alle sine naboruter (unntatt den som nettopp spurte) om å finne en vei videre, osv. På denne måten vil vi til slutt komme til en åpning (hvis det finnes en vei dit).

Kort fortalt er strategien altså å prøve å gå alle veier videre bortsett fra den vi kom fra.

Programmet ditt skal finne alle åpninger du kan nå fra en gitt hvit startrute i en asyklisk labyrint.

(a) Lag en `finn()`-metode i klassen `Rute` med følgende signatur:

```
public void finn(Rute fra) { \* ··· *\ }
```

Parameteren `fra` er med slik at en kan vite hvilken rute kallet kommer fra. Denne metoden skal kalle `finn()`-metoden på alle naboruter (også sorte ruter) unntatt den som er i den retningen kallet kom fra (for da ville vi gått tilbake til der vi nettopp var).

Merk: Det kreves at du implementerer letingen ved hjelp av *polymorfi*, det vil si uten bruk av `instanceof`. Du skal heller ikke bruke rutenes `toString`-metoder i løsningen for å finne utveier. Polymorfi betyr ganske enkelt at subklasser kan implementere metoder med samme *signatur* forskjellig. Metoden `finn()` skal implementeres forskjellig i alle subklassene til `Rute`. Når `Åpning`, `SortRute` og `HvitRute` har hver sin tolkning av metoden `finn()`, trenger vi ikke å bruke `instanceof` eller `toString`.

Skriv metoden `finnUtveiFra (int rad, int kol)` i `Labyrint`-klassen. Denne skal kalle på metoden `finn()` i den ruta som ligger på posisjon `(rad, kol)` i labyrinten.

NB! Ved starten kommer du ikke fra noen annen rute, så alle veier ut fra startruten må prøves.

Hint: La kallet for startruten være `finn(null)` og lag metoden `finn()` slik at om parameteren er `null`, betyr det at alle veier videre skal prøves.

(b) Opprett klassen `Innlevering6` og skriv en `main()`-metode som tar et filnavn som parameter og oppretter en labyrint. Deretter skal programmet gå inn i en kommando-løkke der brukeren oppgir koordinater til én og én startrute og programmet skriver ut koordinatene til alle åpninger fra denne startruten. Dette gjør at man enkelt kan teste åpninger fra mange ruter i samme labyrint.

Under ser du noen eksempler på mulige kjøringer. Din utskrift behøver ikke se akkurat slik ut, men den skal inneholde samme informasjon.

Kjøreeksempel

```
$ java Innlevering6 5x5.labyrint

Slik ser labyrinten ut:

    #    .    #    #    #
    #    .    #    .    #
    .    .    #    #    #
    #    #    #    .    .
    #    .    #    #    #

Skriv inn startkoordinatene <rad> <kolonne> ('-1' for å avslutte)

1 1

Åpninger:

(2,0)

(0,1)

Skriv inn nye startkoordinater ('-1' for å avslutte)

1 3

Åpninger:

Skriv inn nye startkoordinater ('-1' for å avslutte)

2 2

Åpninger:

Du kan ikke starte i en sort rute
```

```
Skriv inn nye startkoordinater ('-1' for å avslutte)

-1
```

Det kan hende at ditt program gir en annen rekkefølge på åpningene fra én bestemt rute. Det er helt OK. Det vesentlige er at programmet finner de samme åpningene.

Oppgave 4 – Valgfrie utvidelser og ekstraoppgaver

Hvis du har tid og lyst, er det mange artige utvidelser du kan gjøre.

(a) Hvis du lurer på i hvilken rekkefølge programmet ditt besøker rutene, kan du lage en enkel utvidelse av programmet ditt. Denne utvidelsen er også nyttig for å sjekke programmet.

Innfør en instansvariabel av typen `int` i alle ruter. Innfør også en statisk tellervariabel som starter på `1`. Hver gang `finn()`-metoden utføres i en rute, kan du hente verdien til tellervariabelen og øke den med én. Verdien lagrer du i instansvariabelen.

Så kan du skrive ut labyrinten etter at du har funnet alle åpninger fra en gitt startrute. I denne utskriften kan du skrive ut verdien av instansvariablen istedenfor punktum. Du skal da forandre virkemåten til metoden `toString()`. Da får du se i hvilken rekkefølge metoden `finn()` blir kalt. Pass på at `toString()`-metoden skriver ut like mange tegn uansett hvilket tall som skal skrives ut (eller om det er . eller #). I eksemplet som er vist her, har vi antatt at instansvariablen har maksimalt tre siffer.

```
$ java Innlevering6 8x9.labyrint

Slik ser labyrinten ut:

    #    #    #    #    #    .    #    #    #
    #    .    .    .    .    .    #    .    #
    #    .    #    #    #    #    #    .    #
    #    .    #    .    .    .    .    .    #
    #    .    #    .    #    #    #    .    #
    #    .    #    .    #    .    #    .    #
    #    .    .    .    .    .    #    .    #
    #    #    #    #    #    #    #    #    #

Skriv inn startkoordinatene <rad> <kolonne> ('-1' for å avslutte)
3 3
Åpninger:
(0,5)

Her er labyrinten slik du gikk gjennom den:

    #    #    #    #    #   16    #    #    #
    #   11   12   13   14   15    #   28    #
    #   10    #    #    #    #    #   27    #
    #    9    #    1   20   21   22   23    #
```

```
    #    8    #    2    #    #    #   24    #
    #    7    #    3    #   19    #   25    #
    #    6    5    4   17   18    #   26    #
    #    #    #    #    #    #    #    #    #

Skriv inn nye koordinater ('-1' for å avslutte)
-1
```

(b) For å kunne finne og skrive ut hele veien fra en startrute til en åpning, må du lagre veien på en eller annen måte. Her er et forslag:

Lag en klasse som heter `Koordinat` som kan inneholde koordinatene til en rute, dvs. to tall `r` og `k`. Utvid `finn()` slik at den tar inn en ekstra parameter som inneholder koordinatene til rutene på den foreløpige veien, slik at finn «husker» veien så langt. Dette skal være en `ArrayList<Koordinat>`. Når du kommer til en ny rute, legger du denne rutens koordinater til `ArrayList`-en før du kaller finn i nabo-rutene. Redefiner også metoden `toString()` i `Koordinat`.

NB! Du må lage en lokal kopi av denne `ArrayList`-en hver gang du kommer til en ny rute. Grunnen til dette er at når du har utforsket en vei, skal du ta fatt på en ny. Da må du ha en «frisk» kopi av veien frem til denne ruten ellers blander programmet sammen den veien du nettopp har utforsket, med den nye veien du nå skal prøve.

Hint: Du kan lage en ny, frisk `ArrayList` med samme innhold som den gamle med: `ArrayList<Koordinat> nySti = new ArrayList<>(sti);`

Når du har funnet en ny utvei, dvs når du er kommet til en åpning, kan utveien frem til denne åpningen legges inn i en `ArrayList` av `ArrayList<Koordinat>`. Du kan lagre listen med utveier som en instansvariabel i `Labyrint` og endre denne når du har funnet en ny utvei. (Husk at hver rute har en referanse til `Labyrint`-en den er en del av.) Endre metoden `finnUtveiFra(int rad, int kol)` i `Labyrint` til å returnere denne listen av alle utveier. Husk at det skal være mulig å kalle denne metoden flere ganger med forskjellige startkoordinater.

Testfilen `13x13.labyrint` har utveiene vist her; se illustrasjonen i Figur 3. Legg merke til at det ikke finnes noen utveier fra `(1, 5)`.

```
$ java Innlevering6 13x13.labyrint

Slik ser labyrinten ut:
 #  #  #  #  #  #  #  #  #  #  #  #  #
 .  .  #  .  .  .  .  .  .  .  .  .  #
 #  .  #  #  #  #  #  #  #  .  #  #  #
 #  .  .  .  #  .  .  .  #  .  #  .  #
 #  #  #  .  #  .  #  .  #  #  #  .  #
 #  .  #  .  #  .  #  .  .  .  .  .  #
 #  .  #  .  #  .  #  #  #  #  #  #  #
 #  .  .  .  #  .  .  .  .  .  .  .  #
 #  .  #  #  #  #  #  #  #  #  #  .  #
 #  .  #  .  .  .  .  .  .  .  .  .  #
 #  .  #  .  #  #  #  #  #  #  #  .  #
```

```
 #  .  .  .  #  .  .  .  .  .  .  .  .
 #  #  #  #  #  #  #  #  #  #  #  #  #


Skriv inn startkoordinater <rad> <kolonne> ('-1' for å avslutte)
3 5
Åpninger:
(1,0)
(11,12)

Utveier:
Løsning nr 1:
(3,5)  (4,5)  (5,5)  (6,5)  (7,5)  (7,6)  (7,7)  (7,8)  (7,9)
↪  (7,10)  (7,11)  (8,11)  (9,11)  (9,10)  (9,9)  (9,8)  (9,7)
↪  (9,6)  (9,5)  (9,4)  (9,3)  (10,3)  (11,3)  (11,2)  (11,1)
↪  (10,1)  (9,1)  (8,1)  (7,1)  (7,2)  (7,3)  (6,3)  (5,3)  (4,3)
↪  (3,3)  (3,2)  (3,1)  (2,1)  (1,1)  (1,0)


Løsning nr 2:
(3,5)  (4,5)  (5,5)  (6,5)  (7,5)  (7,6)  (7,7)  (7,8)  (7,9)
↪  (7,10)  (7,11)  (8,11)  (9,11)  (10,11)  (11,11)  (11,12)


2 løsninger funnet


Skriv inn nye startkoordinater ('-1' for å avslutte)
1 5
Åpninger:


Utveier:
0 løsninger funnet

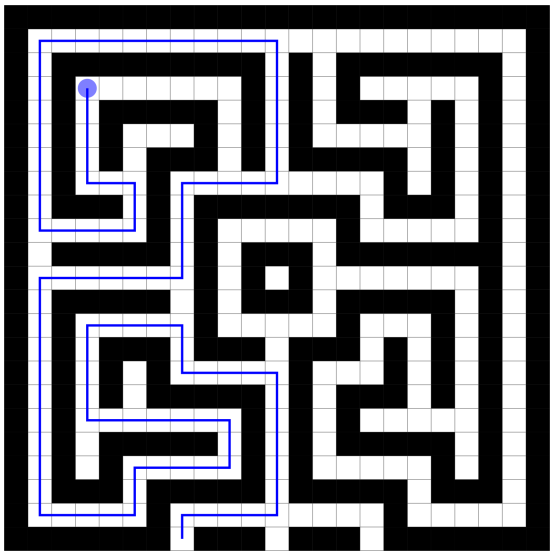

Skriv inn nye startkoordinater ('-1' for å avslutte)
-1
```

(c) For å kunne håndtere sykler, må vi tilføre algoritmen vår et nytt element – vi trenger å merke veien vi har gått, slik at vi kan snu når vi kommer til en rute som allerede er på veien. Dette kan gjøres ved å markere ruter som besøkt med et boolsk flagg, dvs en instansvariabel `besøkt`.

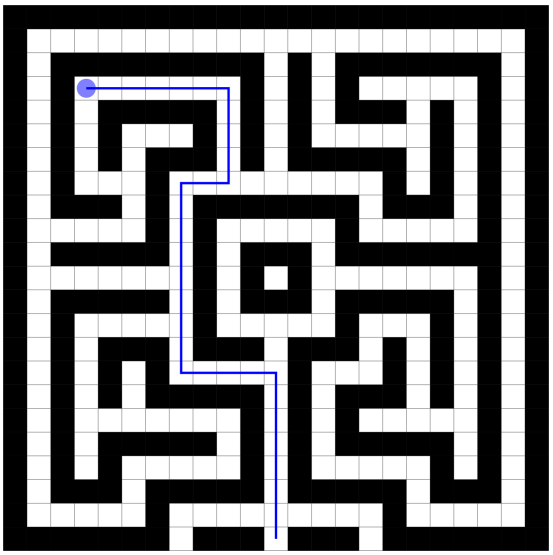
Modifiser koden din slik at vi holder styr på om vi har vært innom rutene under letingen etter en utvei og ikke går gjennom samme rute flere ganger.

(d) I mange labyrinter kan vi finne flere utveier fra et startpunkt. Noen av disse er åpenbart ikke optimale, men de er likevel korrekte. Se på Figur 4 og Figur 5. Utveien i Figur 4 er åpenbart fryktelig ineffektiv – faktisk er den tre ganger så lang som den i Figur 5! Dette kan du finne ut ved å sjekke lengden av utveiene.

Utvid programmet ditt slik at det finner den korteste utveien. Skriv også ut hvor mange utveier som ble funnet.



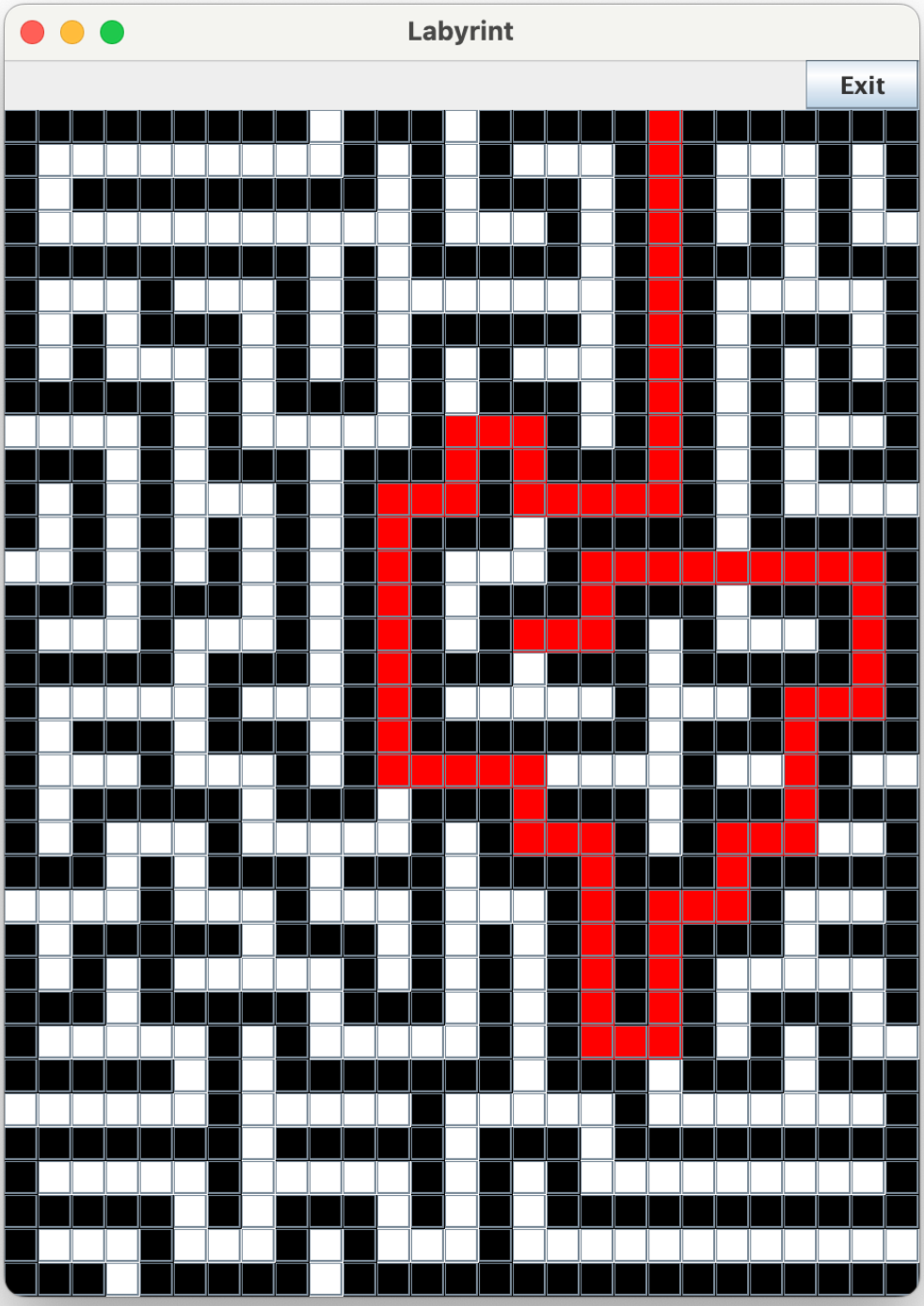
Figur 4: En tilfeldig utvei fra `(3, 3)` i labyrinten i fil `23x23.labyrint`



Figur 5: Den korteste utveien fra `(3, 3)` i labyrinten i fil `23x23.labyrint`

(e) Gi labyrintprogrammet ditt et GUI-grensenitt, for eksempel noe som vist i Figur 6.

Da kan du klikke på en rute og la programmet ditt tegne veien fra den ruten til en åpning (om det finnes en slik vei).



Figur 6: Eksempel på et GUI-grensesnitt