Automatisk robot vedlikehold

Du er ansatt som konsulent for et firma som heter Automatisk robot vedlikehold (ARV). De har mange års erfaring med robot teknologi og styring av roboter. Nylig har ARV fått en ny kontrakt på vedlikehold av solcellepanel på et stort felt med solceller. Solcelle panelene ligger rett ved en søppelfylling og plastposer har en lei tendens til å lande oppå solcellepanelene og dermed produseres det ikke strøm. Firmaet som driver solcellene har laget et system som varsler om solceller som får mindre lys enn andre og dermed må sjekkes for plastposer, men de er nå lei av å fjerne disse posene manuell og har kjøpt inn ARV sine roboter til oppgaven. Styrings programmet til robotene er ferdig testet så de kan ta seg frem hvor som helst på solcelle området. De har også trent opp robotene til å fjerne plastposer og annet på solcellene. Men det tar for lang tid å flytte robotene til rett sted fordi robotene alltid er langt unna der de trengs. De trenger hjelp til noen algoritmer for å velge rett robot til jobben og posisjonere robotene. Firmaet får bonus betaling for hver jobb de fullfører men får trekk for hvert minutt solcellene er dekket av plast.

Online Algoritme

Dette er det vi kaller en Online algoritme, som betyr du må gjøre et valg basert på den informasjonen du har i øyeblikket og når fremtidige jobber kommer oppdager du gjerne at dine vald ikke var optimale.

Heuristikk

Det er ikke krav om å utvikle optimal algoritme for å bevege robotene i denne obligen, så lenge alle jobbene blir gjort er det godkjent. En vanlig algoritme har rett og feil svar mens en heuristikk har mange rette svar men noen av de er bedre enn andre. Vi har laget en test som gir deg poeng for hvor godt robotene gjør det. Det er kun på siste oppgave at disse poengene vil telle.

Følgende deler av programmet er allerede ferdig:

- Location.java – beskriver en lokasjon med to double verdier x og y.
- Job.java – beskriver en Job generert av varslingssystemet som oppdager hvilke solceller som ikke produserer strøm. Job inneholder:
    - Location location - som beskriver hvor solcellepanelet befinner seg
    - En tid som beskriver når det ble oppdaget at solcellepanelet sluttet å produsere strøm.
    - Antall roboter som trengs for jobben.
- Robot.java som beskriver en robot og har metoder for å kontrollere roboten
    - getLocation() – siste kjente lokasjon
    - getJob() – Den jobben roboten er på vei til å gjøre
    - move(Job job) – roboten avbryter andre ordre og går mot jobben og gjør den.
    - move(Location location) – roboten avbryter andre ordre, går mot location og venter.

I tillegg er det en del kode i pakken compulsory.system som styrer robotene og måler hvor lang tid disse bruker på å utføre jobbene. Denne koden trenger dere ikke bry dere om. Hovedoppgaven deres i denne Obligen er å implementere interfacet **IStrategy** og kalle på funksjonene til **Robot** objektene.

Vurdering

Denne Obligen teller 15% av endelig karakter og dere vil få en poengsum fra 0 til 15.

- Kodekvalitet gir 3 poeng
  - Koden skal være ryddig og lesbar
  - Du skal unngå repetisjon av kode og ellers bruke konsepter dekket i INF101
  - Hvis det er ting som er felles for flere strategier bør dette legges i f.eks. en abstrakt klasse.
- Kjøretidsanalyse og forklarende tekst gir 3 poeng
  - Hver funksjon du skriver skal analyseres med kjøretid, du får poeng hvis kjøretiden stemmer med implementasjonen og trekk hvis kjøretiden er feil.
  - Enten skrives dette i et eget dokument eller som kommentarer i koden (se Robot.java hvor vi har gitt kjøretider i kommentarer over metodene)
  - Det er først og fremst Worst case kjøretid i big-O notasjon vi forventer, men om dere ønsker å uttrykke f.eks. forventet kjøretid el. Lignende er alt greit så lenge det er klart beskrevet i readme filen.
  - I readme filen skal dere også skive en forklarende tekst til hver av de tre oppgavene slik at det blir lettere for gruppeleder å forstå hva dere har tenkt (selv om dere ikke helt fikk til å programmere det). Skriv også hvis det var noe som var vanskelig.
- Effektivitet og rett bruk av datastrukturer gir 3 poeng
  - Noen av oppgavene krever bruk av datastrukturer vi har lært i kurset
  - LinkedList, ArrayList, HashMap eller PriorityQueue kan være aktuelle. Tenk over at du har valgt den som du mener passer best.
  - Vi kommer til å se på de metodene du har implementert og vurdere om vi mener du burde gjort det mer effektivt, hvis det er mange steder vi ser du ikke har fått til å implementere optimal algoritme får du trekk. Alle funksjoner trenger ikke å være korrekt men de som påvirker den totale kjøretiden mest bør være gode.
  - Kjøretiden skal uttrykkes best mulig med de 3 parameterene:
    - M – antall jobber i simuleringen (input opp til 1 000 000 forventes)
    - N – antall roboter i simuleringen (input opp til 100 forventes)
    - K – antall roboter som kreves for en jobb (input opp til 10 forventes)
  - Ikke alle disse tre parameterene er relevante for alle metoder, kun bruk de som er relevante.
- Korrekthet av koden (særlig oppgave 1 og 2) gir 3 poeng
  - Oppgave 1 og 2 gir spesifikke instruksjoner du skal klare å implementere. Det er laget JUnit tester som sjekker disse. For å få 3 poeng her må alle JUnit testene passere.
  - Har du fått noen tester til å passere får du noen poeng.
  - Det er også noen tester som ikke er JUnit tester som må passeres (se neste punkt)
- Effektivitet av strategiene (særlig oppgave 3) gir 3 poeng
  - Robotene fikser solsellepanel, for hvert minutt solsellepanelene er ute av funksjon taper de penger. Simuleringen vil gi ut hvor mange minutter solsellepanelene totalt har vært ute av drift før de ble fikset. Jo lavere tall du får jo mer poeng får du. Alle som har fått det bedre enn det Strategiene i oppgave 1 og 2 klarer får minst 1 poeng. Det er ganske greit å få 2 poeng på denne men 3 poeng blir bare gitt til de beste løsningene

Oppgaver

1. I denne oppgaven skal du implementere er random strategi. Vi har allerede laget klar filen du skal implementere : RandomRobotStrategy.java. Du skal velge en random robot som ikke holder på med en annen jobb fra listen av roboter som ble gitt inn til metoden RandomRobotStrategy::registerRobots(List<Robot> robots). Du må sørge for at alle jobber blir gjort, hvis det ikke er nok ledige roboter må du enten ta roboter vekk fra en annen jobb eller vente med jobben til det er nok ledige roboter.

   Beregn kjøretid av de tre metodene registerRobots, registerJob og jobFulfilled. Du kan trenge å beregne kjøretid for andre metoder også for å få rett kjøretid, skriv ned kjøretid også på de hjelpemetodene du trengte å beregne kjøretid av.

2. I denne oppgaven skal du implementere ClosestRobotStrategy.java Denne skal finne de k nærmeste robotene som er ledige og sende disse til Jobben.

   Sjekk at testene… passerer.

   Kjør TestClient.java og studer poengsummen på de to forskjellige strategiene.

   Gjør denne Strategien det bedre eller dårligere enn random strategien? Hva tror du er grunnen til at denne strategien gjør det dårligere på noen tester? Skriv svar i readme filen.

   Beregn kjøretid av de tre metodene registerRobots, registerJob og jobFulfilled. Du kan trenge å beregne kjøretid for andre metoder også for å få rett kjøretid, skriv ned kjøretid også på de hjelpemetodene du trengte å beregne kjøretid av.

3. Vi oppdaget at random strategien noen ganger gjør det bedre enn closest, kan du tenke det et senario der closest gjør en veldig dårlig jobb i forhold til hva som er mulig?

   Lag en ny strategi som implementerer IStrategy og legg til denne strategien i TestClient.java. Kjør TestClient og se om du klarer å gjøre det bedre enn de to foregående strategiene.

   Beskriv ideene dine i readme filen og forklar hvorfor du tror dette vil forbedre robotene.

   Beregn kjøretid av de tre metodene registerRobots, registerJob og jobFulfilled. Du kan trenge å beregne kjøretid for andre metoder også for å få rett kjøretid, skriv ned kjøretid også på de hjelpemetodene du trengte å beregne kjøretid av.

Testing

Vi har gitt dere noen tester som kjøres av TestClient, vi kommer til å teste på en litt annen input enn den dere har fått utdelt. Det er derfor viktig at dine strategier virker både på de test casene du har fått utdelt, men også generelt være god i de andre situasjonene vi kommer til å teste.