ROBOCODE INNLEVERING DOC

Av Ole Vedeler & Øivind Jorfald

**Dette er dokumentasjonen til vår robocode innlevering. Her står det noe om hvilke valg vi tok, hva som funket og ikke funket, og forbedringer vi ville ha gjort hvis vi hadde hatt bedre tid. Vi har delt det opp i 3 seksjoner: Finite-state machines der vi snakker om hvorfor vi valgte det vi valgte; styrings algoritmer der vi snakker om hvilke og hvorfor vi valgte de forskjellige algoritmene; Forbedringer til koden hvis vi hadde hatt bedre tid.**

**FSM**

Vi sto mellom 3 finite-state machines: Active state generates, Hierarchical og child-state compete. Vi valgte å ikke bruke child-state compete fordi vi mener det gir litt for dårlig ytelse, og passer ikke så bra inn her. Vi bestemte oss da for å bruke Active state generates fordi det ikke blir like uoversiktlig som hierarchical FSM og lettere å implementere. Vi prøvde oss først på en FSM som brukte klasser, men etter å ha kodet i en stund fant vi ut at det ble vanskelig bruke Advancedrobot sine metoder uten å sende med selve robot objektet, og vi ville ikke skape kryss koblinger mellom klassene, så vi bestemte oss for å hardkode det med enums istedenfor. Etter å ha hvert i timen og så at faglærer sendte med robot objektet inn i klassene bestemte vi oss for å prøve en gang til, så vi prøvde å implementere en FSM ved hjelp av boken Artificial Intelligence for games sin psudokode. Men tydeligvis er ikke denne boken den beste å hoppe rett inn i, og har lite referanser til hvor bak i boken jeg skal finne denne informasjonen. Så etter å ha brukt en dag på dette ga vi opp FSM klasser og brukte den gamle hardkodede enum-baserte FSM vår.

Vi valgte å dele statene opp i 2 forskjellige FSM, den ene styrte våpenet og radaren, mens den andre styrer kjøringen. Kjøringen har 3 states: RAM, ESCAPE og AVOID. Ram kjører inn i fienden hvis du har mer energi en dem, ESCAPE prøver å komme seg vekk og holde seg på avstand, hvis den har lite energi. Avoid prøver å ikke bli skutt med forskjellige manøver. Våpen og radar har også 3 states: ATTACK, SAVEENERGY og SCAN. Attack brukes hvis den har mer energi en fienden og kan bruker mer kraft til kulene. SaveEnergy har litt svakere kuler for å spare på energien for å ikke komme så mye under og prøve å få fienden til å bruke opp sitt. Scan slår inn i begynnelsen og hvis roboten mister fienden på radaren.

**Styrings algoritmer**

Vi tenkte på mange forskjellige styrings algoritmer og kom fram til: Flee; Evade; Pursuit; Seek; Wall avoidence; offset pursuit. I tillegg lagde vi vår egen for sikting av våpenet, vi var ikke sikker på om den skulle være inne sammen med styrings algoritmene, men kom fram til at den passet der fordi det er fortsatt styring av roboten.

* Vi valgte å ikke bruke Arrive fordi vi syntes pursuit var mer egnet for å krasje i fienden og tenker mer fart gir bedre kollisjon.
* Vi valgte å bruke seek fordi pursuit kunne bruke den til å styre mot det forventede punktet, og hvis vi fant ut at vi ville bruke den senere så hadde vi da allerede implementert den.
* Vi valgte pursuit fordi den var best til å stoppe fienden og ikke bare krasje inn i bakdelen av bilen som seek eller arrive ville ha gjort.
* Vi bruker flee for å stikke av fra fienden når vi har mindre liv.
* Vi bruker evade for å unngå skudd ved å sette seg 90 grader på fienden sin heading, altså slik at når fienden skyter kan du bevege deg enten fram eller bakover for å unngå kulene
* Vi bruker ikke Wander fordi vi trenger litt mer tilfeldige uforutsigbare bevegelser istedenfor fine buer for å unngå kuler.
* Vi bruker ikke obstacle avoidance fordi det ikke er noen/noe å krasje i annet en veggen og fienden, og veggen bruker vi wall avoidence. Vi kunne ha brukt den til å unngå fienden, men vi er en angrepsrobot så vi så ikke helt trengselen for det.
* Vi bruker wall avoidence for å unngå å krasje inn i vegger.
* Det er ingen grunn til å bruke interpose, fordi vi ikke har noen å sette oss foran
* Er ingenting å gjemme seg bak, så er ikke vits å implementer hide
* Vi synes path following blir litt for åpenbart, og lett å slå. Vi velger heller å ta en litt mer tilfeldig vei å gå.
* Vi valgte å bruke offset pursuit for å kunne holde oss på en konkret avstand fra fienden. Men vi holder oss ikke til en konkret posisjon i forhold til fienden som man egentlig skal.

**Forbedringer**

* Få en FSM med klasser til å fungere
* Få kule sikting til å treffe oftere
* Lage statistikk om kuler som treffer og kuler som bommer, og justere sikte etter det
* Fikse slik at wall avoidence fungerer skikkelig
* Gjøre alt mer objekt-orientert
* Legge til flere states, som f. eks lockradar