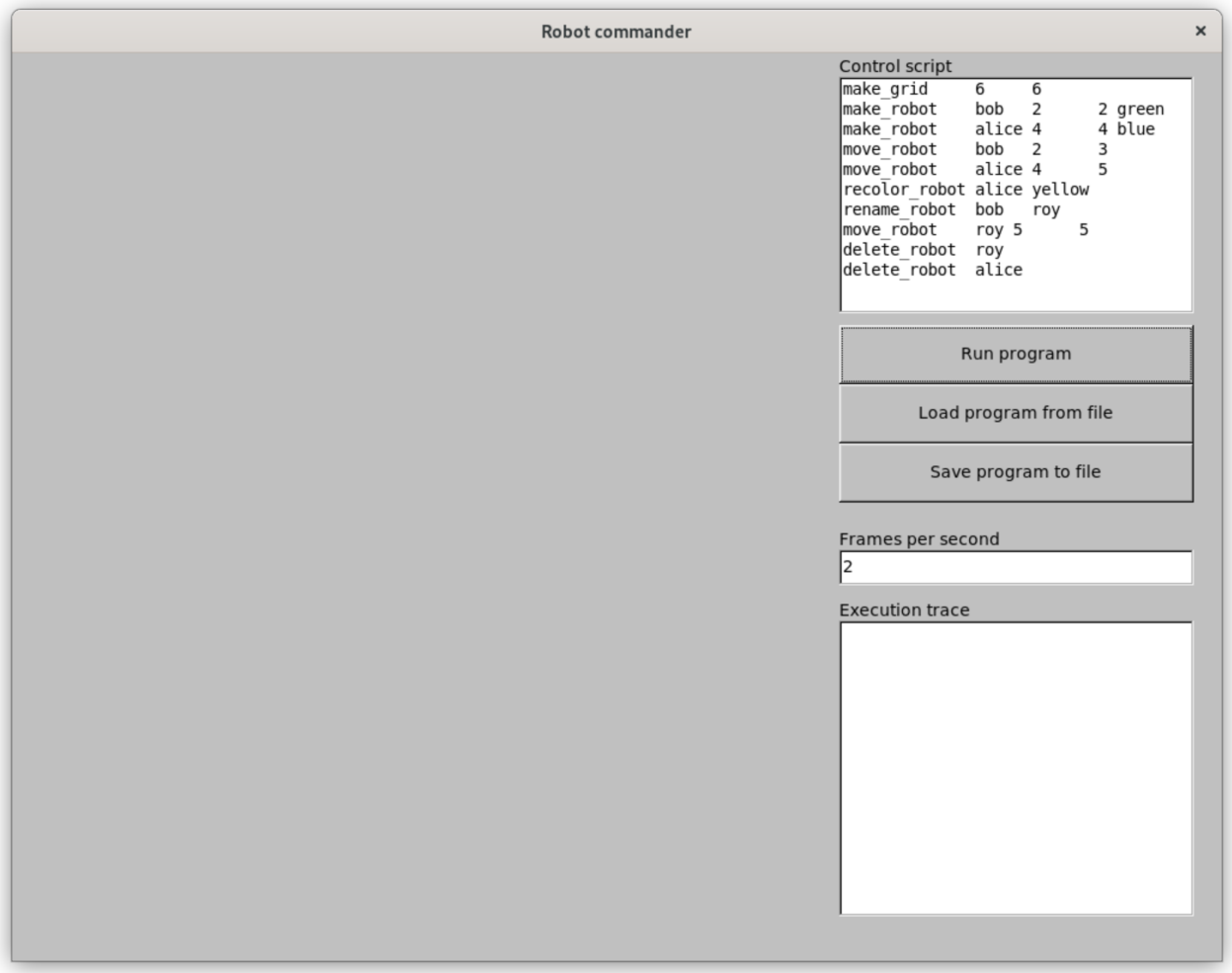
Part III: RobotCommander

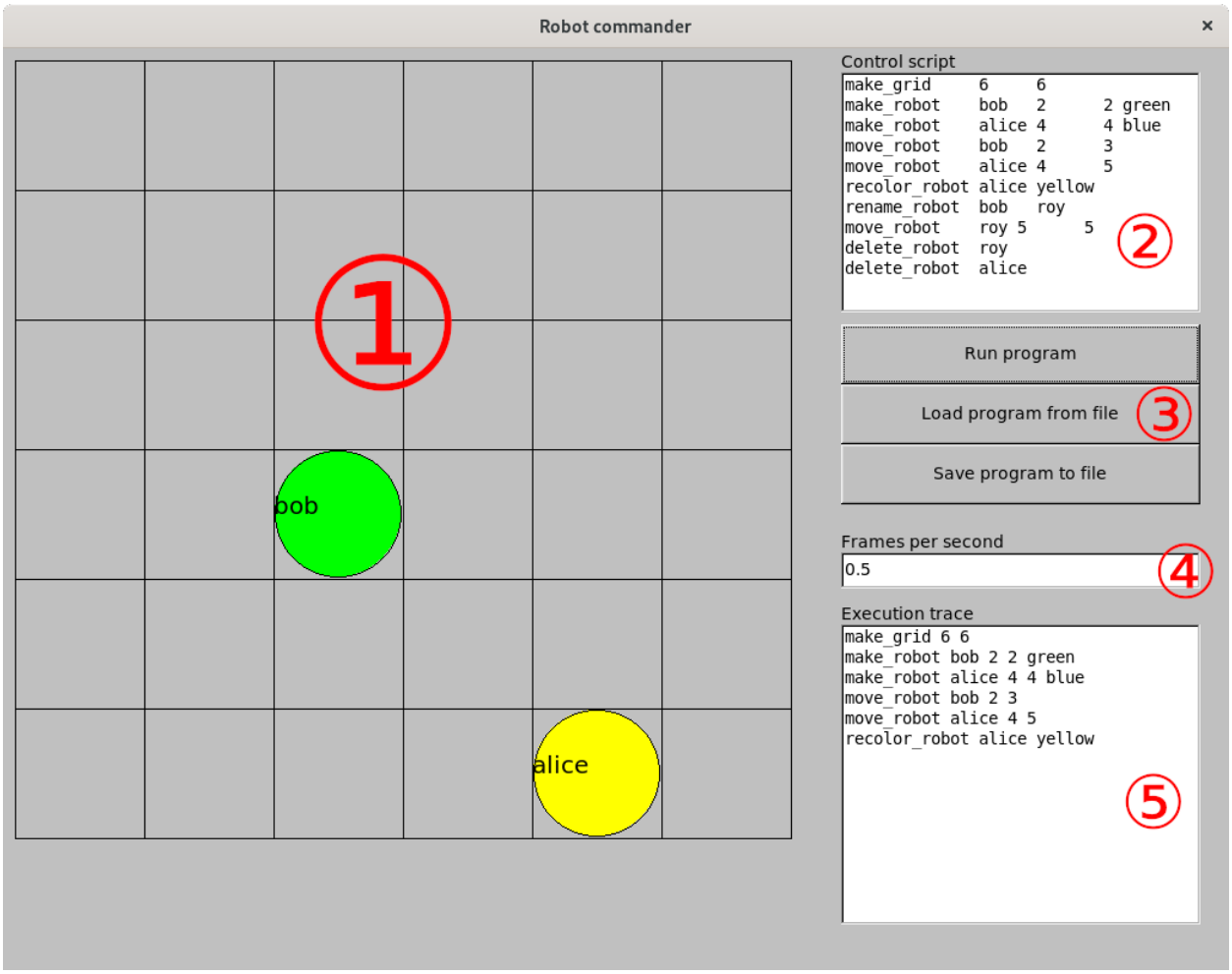
Maksimal skår for del 3 er 180 poeng.

3.1 Introduksjon

RobotCommander er ein applikasjon som utfører enkle kommandoar (skrive på eit bestemt, enkelt format) som flyttar og endrar på "robotar" i eit 2D-univers. Resultatet av kommandoane blir vist visuelt ved å visa robotane i eit 2D rutenett med eit justerbart tal rader og kolonnar. I `zip`-fila handout finn du eit køyrbart applikasjonsskjelett som manglar nokre viktige delar av den ønska funksjonaliteten til RobotCommander. Denne eksamensoppgåva går ut på å implementera desse manglande bitane slik at RobotCommander fungerer som ønska. Du vil leiast gjennom denne prosessen ved hjelp av alle deloppgåvene i denne delen av eksamen. Under finn du ein grundig introduksjon til og forklaring av korleis RobotCommander fungerer.



Figur 1: Skjermbilde av applikasjonen slik han ser ut når du køyrer handout-koden (utan å ha endra på handout-koden). **Merk:** Programmet som ligg inne i "Control script"-vindauget som kjem når du køyrer den utdelte koden er *ikkje* kjørbart før du har implementert koden som blir kravd for kvar av instruksjonane.



Figur 2: Skjermbilde av at den ferdige applikasjonen køyrer eit "control script". Sjå Avsnitt 3.2 for ei forklaring av dei ulike GUI elementa markert med raude tal med sirklar rundt.

3.2 Slik fungerer RobotCommander

I denne delen finn du forklaring på korleis RobotCommander skal fungera når han er ferdig implementert. Tala med sirkel rundt i denne delen refererer til skjermbildet de finn i Figur 2. Merk at dette skjermbildet viser korleis applikasjonen verkar når den er ferdig, det vil seia *etter* at de har løyst heile del 3 av denne eksamenen. For eit skjermbilde av korleis han ser ut når ein køyrer den utdelte koden, sjå Figur 1. Hovuddelen av applikasjonen består av rutenettet ① som representerer universet robotane eksisterer i. Robotane blir visualiserte basert på posisjon, farge og namn, slik de ser i figuren. I figur 2 er roboten kalla bob grøn og står på plass $(2, 3)$, og roboten kalla alice er gul og står på plass $(4, 5)$. Kontrollskriptet ("Control script"), sjå nærare forklaring i Avsnitt 3.3, er skrive på eit redigerbart tekstområde ②. Knappane ③ blir brukte til å køyra kontrollskriptet (*Run program*), lasta inn eit eksisterande program frå ein fil til "Control script"-tekstområdet (*Load program from file*), og til å lagra programmet som står skrive i "Control script"-tekstområdet til ei fil (*Save program to file*). Verdien i tekstområde ④ blir brukt til å kontrollera kor raskt RobotCommander køyrer kontrollskriptet. Ettersom programmet utfører kommandoane blir desse vist i sporingsfeltet ("Execution trace") ⑤. Dette gjer det lettare å debugge då du kan sjå kva kommando som nettopp har vorte utført for å forstå kor feilen truleg oppstod.

3.3 Korleis bruka RobotCommander

Eigenskapane og posisjonen til robotane blir kontrollert ved hjelp av kontrollskript (*control scripts*). Eit kontrollskript er (i denne samanhengen) ein sekvens med instruksjonar som kan laga, sletta, endra og flytta på robotar i rutenettet. Kontrollskriptet er strukturert slik at det er ein kommando/instruksjon per linje. Desse instruksjonane blir alltid gitte på same format: namnet på instruksjonen, følgd av null eller fleire argument. La oss sjå på eit eksempel på eit slikt kontrollskript.

Eksempelprogram:

```
make_grid     6     6
make_robot    bob   2      2 green
make_robot    alice 4      4 blue
move_robot    bob   2      3
move_robot    alice 4      5
recolor_robot alice yellow
rename_robot  bob   roy
move_robot    roy 5      5
delete_robot  roy
delete_robot  alice
```

Først blir eit nytt "univers" laga ved å bruka `make_grid` instruksjonen. Tala etter `make_grid`, 6 og 6, er argumenta som blir sende inn til denne instruksjonen. I dette tilfellet representerer dei tal rader og kolonnar, slik at vi opprettar eit rutenett med 6 rader og 6 kolonnar. Dei to følgjande `make_robot` instruksjonane lagar robotane bob og alice, med startkoordinatar, $(2,2)$ for bob og $(4,4)$ for alice, og fargane grøn (bob) og blå (alice). `move_robot` instruksjonane flyttar deretter dei to robotane til nye posisjonar i rutenettet, høvesvis $(2,3)$ for bob og $(4,5)$ for alice. Merk at rutenettet er nullindeksert slik at posisjon $(0,0)$ er øvste rute til venstre og $(5,5)$ er nedste rute til høgre i dette rutenettet. `recolor_robot` kommandoen endrar fargen til alice til gul. Deretter endrar bob namn til roy ved hjelp av `rename_robot` instruksjonen. Merk at det ikkje finst nokon robot kalla bob etter denne endringa, så kvar referanse til dette namnet vil derfor vera ugyldig. For å flytta roboten som før heit bob må ein derfor bruka det nye namnet roy i staden. Avslutningsvis blir dei to robotane roy og alice sletta ved å bruka `delete_robot` instruksjonen. Alternativt kunne ein brukt `clear_robots` (denne tar ikkje inn nokon argument) for å sletta alle robotane i rutenettet med ein enkelt kommando.

Merk at instruksjonane vil feila dersom argumenta er ugyldige. Døme på ugyldige argument er mellom anna:

- koordinatar utanfor grensene til rutenettet
- ikkje-eksisterande fargar
- bruk av ugyldige robotnamn, t.d. namn som ikkje er i bruk

Vi vil sjå nærare på desse avgrensingane etterpå når du skal implementera funksjonar for å sjekka argumentas gyldigheit.

Oversyn over instruksjonane/kommandoane som skal støttast i RobotCommander finn du i Tabell 1. Denne kan vera eit nyttig oppslagsverk i dei seinare deloppgåvene.

Vi har lagt ved ein kort video (`robotcommander_video.mp4`, utan lyd) i den utdelte zip-filen som viser korleis ei køyring av kontrollskriptet som vart vist og forklart i avsnitt 3.3 i ferdig implementert RobotCommander ser ut.

Korleis svara på del 3?

Kvar oppgåve i del 3 har ein tilhøyrande unik kode for å gjera det lettare å finna fram til kor du skal skriva svaret. Koden er på formata `<tegn><siffer>` (TS), eksempelvis A1, A2 og G1. Eit oversyn over koplinga

Tabell 1: Oversyn over instruksjonane/kommandoane som kan brukast på robotane.

Instruksjon/kommando	Skildring
`make_grid <rows> <cols>`	Lagar eit nytt rutenett med eit bestemt tal rader og kolonnar. Denne kommandoen ryddar opp i alle eksisterande tilstandar og slettar alle roboter.
`make_robot <name> <x_pos> <y_pos> <color>`	Lagar ein robot kalla `name` på rutenettposisjon (`x_pos`, `y_pos`) med farge `color`
`clear_robots`	Fjernar alle robotar frå rutenettet.
`move_robot <name> <x_pos> <y_pos>`	Flyttar roboten kalla `name` til rutenettposisjonen (`x_pos`, `y_pos`)
`recolor_robot <name> <color>`	Endrar fargen til roboten kalla `name` til `color`
`rename_robot <name> <new_name>`	Endrar namnet til roboten kalla `name`. Endrer frå `name` til `new_name`
`delete_robot <name>`	Slettar roboten kalla `name`

mellom bokstav og fil er gitt i tabell 2. I robot_grid.cpp, og dei andre filene du skal løyse oppgåver i, vil du for kvar oppgåve finna to kommentarar som definerer høvesvis byrjinga og slutten av koden du skal føra inn. Kommentarane er på formatet: `// BEGIN: TS` og `// END: TS`.

Det er veldig viktig at alle svara dine er skrivne mellom slike kommentar-par, for å støtta sensurmekanikken vår. Viss det allereie er skrive noko kode *mellom* BEGIN- og END-kommentarane i filene du har fått utdelt, så kan, og ofte bør, du erstatta den koden med din eigen implementasjon med mindre anna er spesifisert i oppgåveskildringa. All kode som står *utanfor* BEGIN- og END-kommentarane SKAL de la stå.

Til dømes, for oppgåve G1 ser du følgjande kode i utdelte `robot_grid.cpp`

```
1  void RobotGrid::draw_grid_lines()
2  {
3    // BEGIN: G1
4    //
5    // Write your answer to assignment G1 here, between the // BEGIN: G1
6    // and // END: G1 comments. You should remove any code that is
7    // already there and replace it with your own.
8
9    // END: G1
10 }
```

Etter at du har implementert løysinga di, bør du enda opp med følgjande i staden:

Tabell 2: Oversyn over koplinga mellom oppgåve-bokstav og filnamn.

Bokstav	Fil
A	`application.cpp`
S	`interpreter.cpp`
G	`robot_grid.cpp`

```
void RobotGrid::draw_grid_lines()
{
  // BEGIN: G1
  //
  // Write your answer to assignment G1 here, between the // BEGIN: G1
  // and // END: G1 comments. You should remove any code that is
  // already there and replace it with your own.

  /* Din kode her */

  // END: G1
}
```

Merk at BEGIN- og END-kommentarane **IKKJE skal fjernast**.

Til slutt, viss du synest nokon av oppgåvene er uklare, oppgi korleis du tolkar dei og dei antagelsene du må gjera som kommentarar i den koden du sender inn.

Før du startar må du sjekka at den (umodifiserte) utdelte koden køyrer utan problem. Du skal sjå det same vindauget som i Figur 1. Når du har sjekka at alt fungerer som det skal er du klar til å starta programmering av svara dine.

3.4 Kor i koden finn du oppgåvene?

Kvar oppgåve har ein unik kode bestående av an bokstav følgd av eit tal. Bokstaven indikerer kva fil oppgåva skal løysast i. Sjå Tabell 2 for eit oversyn over bokstavar og filnamn. Eksempelvis finn du oppgåve A1 i fila `application.cpp`

3.5 Oppgåvene:

Lage og modifisera robotar (90 poeng)

I denne fyste delen av oppgåvene skal du implementera kode for å laga, modifisera og teikna robotene. Men fyst har vi ei lita forklaring av visualiseringsmodellen til RobotCommander og korleis robotene blir representert.

Ein robot er representert av structen `Robot` som er definert under. Denne er definert i fila `robot.h` i den utdelte koden. Alle roboter i universet er instansar av denne structen.

```
struct Robot {
  Robot(string name, Point pos, Graph_lib::Color color);

  string name;
  Point pos;
  Graph_lib::Color color;
};
```

Koden for å manipulera og teikna robotene ligg i fila `robot_grid.cpp` og i klassa `RobotGrid`. Denne klassa har eit map `robots` definert som

```
map<string, unique_ptr<Robot>> robots;
```

som koblar namnet til robotene med instansar av structen `Robot`.

Funksjonane `RobotGrid::draw_grid_lines()` og `RobotGrid::draw_robots()`, definert i `robot_grid.cpp`, teiknar delar av den grafiske representasjonen til universet, nærare bestemt linjene i rutenettet og robotene. Sjå Figur 2 ① for eit døme på korleis dette vil kunne sjå ut. Desse funksjonsne teiknar heile universet på nytt kvar gong dei kallas. Du slipp derfor å ta omsyn til tilstanda til visualiseringen. Du skal implementera desse funksjonane i oppgåve G1 og G4. Denne måten å designe programmet på gjer at alle endringar i universet blir vist i den grafiske representasjonen umiddelbart, utan at ein eksplisitt må kalla til dømes ein GUI "update"-funksjon.

1. (10 points) **G1: Teikn linjene i rutenettet**

   Implementer koden for å teikne linjene i rutenettet, altså `RobotGrid::draw_grid_lines()`-funksjonen. For å løyse denne oppgåva bør du bruka `window.draw_line()` funksjonen for å teikne linjene og følgjande medlemsvariablar i `RobotGrid`-klassa for å bestemme utsjånaden til rutenettet:

   - `x_pos` og `y_pos`: koordinatane til øvre venstre hjørne i rutenettet
   - `w_size` og `h_size`: breidda og høgda til rutenettet
   - `cell_width` og `cell_height`: breidda og høgda til kvar celle i rutenettet.

2. (10 points) **G2: Finn midten av ei celle i rutenettet**

   Skriv funksjonen `RobotGrid::get_grid_cell_center_coord()` som, gitt `x` og `y` koordinatane til ei rutenett-celle, returnerar skjerm-koordinatane til sentrumet av cella. Til dømes vil sentrum i rutenett-celle $(1, 1)$ kunne vera $(150, 150)$ pikslar frå øvre venstre hjørne i applikajonsvindauget (avhengig av breidda og høgda etc til cella). Du kan rekna ut dette vha. dei same medlemsvariablane som vi tipsa om i oppgåve G1.

   ```
      |     |
   ---+-----+---
      |  x  |
   ---+-----+---
      |     |
   ```

   For å illustrere, skissa over viser eit rutenett. Funksjonen skal returnere skjermkoordinatane til ei vilkårleg celle `x`. **Merk:** Du kan anta at koordinatane som vorte sendt til funksjonen er innanfor grensane til rutenettet. Vi handterer unntak i ei seinare oppgåve.

3. (10 points) **G3: Lag ein robot**

   Implementer koden for å laga ein ny robot, `RobotGrid::make_robot()`-funksjonen. Du gjer dette ved å laga ein instans av `Robot`-structen (definert i `robot.h`) og leggje han til i `robots`-mapet i `RobotGrid`-klassa. Du treng ikkje å sjekke om posisjonen og namnet er gyldig, dette blir handtert i ei seinare oppgåve. Hugs at `robots` er eit map frå `string` til `unique_ptr<Robot>`.

   Du må gjere oppgåve G4 (teikne robot) og oppdatera delar av koden beskrive i oppgåve S2 for å testa denne funksjonen. Merk at robotane ikkje får korrekt farge før oppgåve S1 har vorte løyst.

4. (10 points) **G4: Teikning av robotane**

   Skriv kode for å teikna robotane i rutenettet, `RobotGrid::draw_robots()`-funksjonen. Ein robot blir representert av ein sirkel som vert teikna innanfor grensane til rutenettcella roboten er plassert i. Sirkelen er fylt med fargen til roboten, og namnet til roboten skal stå skrive oppå sirkelen med startpunkt midt på venstre side av cella. Du treng ikkje sjekke om posisjonen og namnet er gyldig, dette vorte handtert i ei seinare oppgåve.

For å vite kor du skal plassera robot-sirklane kan det vera lurt å bruka `get_grid_cell_center_coord()`-funksjonen du nettopp har implementert. Funksjonen `get_grid_cell_edge_coord()`, som er ein del av den utdelte koden, kan også vera nyttig i denne oppgåva.

Du må oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

5. (10 points) **G5: Slett ein robot**

   Implementer koden for å slette ein robot fra rutenettet, `RobotGrid::delete_robot()`-funksjonen. Hugs at alle robotane i rutenettet er lista i `robots` map-et.

   Du må oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

6. (10 points) **G6: Flytt ein robot**

   Implementer koden for å flytta på ein robot, altså funksjon `RobotGrid::move_robot()`. Gjer dette ved å endra på den korresponderande instansen av `Robot` i `robots`- map-et.

   Du må oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

7. (10 points) **G7: Endre fargen til ein robot**

   Implementer kode for å endre fargen på ein robot, mao. funksjon `RobotGrid::recolor_robot()`.

   Du må løyse oppgåve S1 og oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

8. (10 points) **G8: Tøm rutenettet**

   Implementer koden for å fjerna alle robotane frå rutenettet, altså funksjon `RobotGrid::clear_robot()`.

   Du må oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

9. (10 points) **G9: Gi roboten nytt namn**

   Implementer koden for å gi ein robot nytt namn, i.e. `RobotGrid::rename_robot()`-funksjonen. Hugs å endra nøkkelen til roboten i `robots` map-et!

   Du må oppdatera delar av koden beskrive i oppgåve S2 for å kunne teste denne funksjonen.

Implementering av instruksjonstolkeren (20 poeng)

10. (10 points) **S1: Frå string til Color**

    Denne funksjonen tar inn namnet på ein farge som ein `string` og returnerar `Graph_lib` Color sin versjon av denne fargen. Denne funksjonen er dermed nødvendig for å kunne konvertera fargenamnene som blir gitt som tekststreng-argumentar til dei ulike instruksjonane til datatypen Color som koden treng for å kunne fargeleggje robotane i rett farge. I den utdelte koden finn du map-et `color_map` i fila `interpreter.h`. Denne inneheld alle gyldige koplingar mellom fargenamn og farger.

    Di oppgåve er å skriva funksjonen `Interpreter::get_color()` som finn Color-versjonen av fargen ved hjelp av `color_map`-map-et. Dersom fargen ikkje finst i map-et skal det bli kasta eit unntak med ein beskrivande feilmelding.

11. (10 points) **S2: Instruksjonstolkeren**

    Instruksjonstolke-funksjonen `Interpreter::execute_instruction()` les ein instruksjon (som beskrive i Tabell 1) og kallar den korresponderande funksjonen i `RobotGrid`-klassa for å utføra den ønska handlinga på universet.

    For å hjelpa deg i gang har vi gitt strukturen til funksjonen og implemenert `make_grid` instruksjonen. Di oppgåve er å implementera korleis alle dei andre instruksjonane skal bli handtert. Alle instruksjonane

kan bli implementert med same metode/teknikk som den vi brukte for å implementera `make_grid`-instruksjonen. Det er opp til deg om du vil følgje denne strukturen eller ikkje. Dersom du er usikker på kva funksjon frå `RobotGrid`-klassa du skal kalla på kan det vera lurt å sjå etter ein funksjon som har same namn som instruksjonen.

Denne funksjonen, altså `Interpreter::execute_instruction()`, blir kalla ein gong for kvar einaste instruksjon i programmet. Dette inneber at for det følgjande kontrollskriptet

```
make_robot ayesha 1 1 blue
move_robot ayesha 2 4
```

blir denne funksjonen kalla to gongar, ein gong for kvar instruksjon. `instruction`-argumentet til funksjonen er ein `istringstream` som inneheld ei linje frå kontrollskriptet. Med andre ord vil dei to funksjonskalla på `execute_instruction` som er vist under bli utført i koden når kommandoane over blir køyrt i GUI-vindauget.

```
execute_instruction(istringstream("make_robot ayesha 1 1 blue"));
execute_instruction(istringstream("move_robot ayesha 2 4"));
```

Sikre at kun lovlege instruksjonar blir køyrt (40 poeng)

I dei neste oppgåvene skal du implementera funksjonar om sjekkar om ein instruksjon og parametrane til funksjonen er gyldige eller ikkje. Universet må til kvar tid oppretthalde ei rekke eigenskapar, sjå lista under. Ein instruksjon er ugyldig dersom vi oppdagar at eigenskapane til universet ikkje vil bli oppretthaldt dersom intruksjonen hadde blitt utført. Eigenskapane til universet er:

- Alle robotnamn må vera unike.
- Det kan *ikkje* stå fleire robotar i same rute/celle i rutenettet.
- Alle robotar må stå på ein gyldig posisjon i rutenettet.
- Instruksjonar kan berre referera til/bli brukt på eksisterande robotar.
- Alle robotar må ha ein gyldig farge.

12. (10 points) **G10: Sjekk av koordinatane**

    Denne funksjonen, `RobotGrid::check_coord_bounds()`, sjekkar om koordinatane som har blitt sendt inn til funksjonen som punktet `p` er gyldige eller ikkje. Gyldige koordinatar er: $0 <= x < cols$ og $0 <= y < rows$. Funksjonen skal kaste eit unntak med ein beskrivande feilmelding dersom koordinatane er ugyldige. Dersom dei er gyldige skal ikkje funksjonen gjere noko.

13. (10 points) **G11: Sjekk om robotnamnet er ledig**

    Denne funksjonen, `RobotGrid::check_name_available()`, sjekkar om `name` er ledig (og dermed kan bli brukt av ein robot). Dette er tilfellet dersom parameteren `name` *IKKJE* er ein nøkkel i `robots` map-et. Kast eit unntak med ein beskrivande feilmelding dersom namnet allereie er i bruk.

14. (10 points) **G12: Sjekk om roboten eksisterar**

    Denne funksjonen, `RobotGrid::check_name_exists()`, sjekkar om ein robot kalla `name` eksisterar. Dette er tilfellet dersom `name` er ein nøkkel i `robots` map-et. Dersom namnet *IKKJE* er i bruk skal det bli kasta eit unntak med ein beskrivande feilmelding.

15. (10 points) **G13: Sjekk om ei rute i rutenettet er ledig**

    Denne funksjonen, `RobotGrid::check_coord_empty()`, sjekkar om det er ein robot i ruta med koordinatane gitt som punktet `p`. Parameteren `is_moving` er `true` dersom funksjonen blir kalla når det er

instruksjonen `move_robot` som blir forsøkt utført, og `false` dersom funksjonen blir kalla når det er instruksjonen `make_robot` som blir forsøkt utført. `name` er namnet på roboten som blir flytta eller laga. Skilnaden mellom flytting og laging er viktig da det er tillatt å flytte ein robot til seg sjølv. Det vil seie at programmet:

```
make_robot tao 1 1 blue
move_robot tao 1 1
```

er gyldig, medan programmet:

```
make_robot tao 1 1 blue
make_robot fatima 1 1 green
```

ikkje er det.

Funksjonen skal kaste eit unntak med ein beskrivande feilmelding dersom ruta/cella allereie inneheld ein anna robot. Hugs på at det er lov å flytta til seg sjølv som beskrive over.

Fil I/O og verdi-konvertering (30 poeng)

I den siste delen av denne eksamenen skal vi implementera dei siste manglande delane av brukergrensesnittet: Laste inn frå og lagre program til filer, og få programmet til å utføra instruksjonane i tempoet gitt av verdien i "Frames per second"-tekstfeltet.

16. (10 points) **A1: Laste inn eit program frå ei fil**

    Implementer funksjonen `Application::load_program()` for å lesa innhaldet i ei fil gitt ved parameteren `file_name` og returner innhaldet i fila som ein string. Pass på å inkludera whitespace (linjeskift, mellomrom etc) frå fila i stringen. Dette gjer det muleg å lasta inn eit program frå ei fil til `Control script`-vindauget i applikasjonen. Eit par test-program har blitt inkludert i den utdelte koden. Du kan bruke desse til å testa funksjonen dersom du ikkje har tid/lyst til å laga dine eigne testfiler. Eventuelle feil som oppstår når fila blir forsøkt åpna skal bli handtert på ein fornuftig måte.

17. (10 points) **A2: Lagre eit program i ei fil**

    Implementer funksjonen `Application::save_program()` for å skrive innhaldet i string-parameteren `contents` til ei fil med filnamnet gitt av parameteren `file_name`. Dette gjer det muleg å lagra det noverande skriptet i `Control script`-vindauget til ei fil. Eventuelle feil som oppstår når fila blir forsøkt åpna skal bli handtert på ein fornuftig måte.

18. (10 points) **A3: Konvertera `string` til `double`**

    Implementer funksjonen `Application::get_fps()` slik at den returnerar stringen som blir tatt inn som parameteren `fps` som ein `double`. Dersom verdien til `fps` ikkje kan bli gjort om til ein gyldig `double` skal det bli kasta eit unntak med ein beskrivande feilmelding.