

Del 3: Bondesjakk

Del 3 av eksamen er programmeringsoppgaver. Denne delen inneholder **11 oppgaver** som til sammen gir en maksimal poengsum på 180 poeng og teller **ca. 60 %** av eksamen. Hver oppgave gir *maksimal poengsum* på **5 - 30 poeng** avhengig av vanskelighetsgrad og arbeidsmengde.

Den utdelte koden inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med forhåndskodede deler og en full beskrivelse av oppgavene i del 3 som en PDF. Etter å ha lastet ned koden står du fritt til å bruke et utviklingsmiljø (VS Code) for å jobbe med oppgavene.

Du kan laste ned .zip-filen med den utdelte koden fra Inspera. I neste seksjon finner du en beskrivelse av hvordan du gjør dette. Før du leverer eksamen, må du huske å laste opp en .zip-fil med den utdelte koden og endringene du har gjort når du har løst oppgavene.

Nedlasting og oppsett av den utdelte koden

De seks stegene under beskriver hvordan du kan laste ned og sette opp den utdelte koden. En bildebeskrivelse av stegene 1 til 5 kan du se i Figur 10.

1. Last ned .zip-filen fra Inspera
2. Lokaliser filen i File Explorer (Downloads-mappen)
3. Pakk ut .zip-filen ved å høyreklikke på filen og velg:
7-Zip → Extract here
4. Kopier mappen som dukker opp til C:/Temp
5. Åpne mappen som ligger i C:/Temp i VS Code ved å velge:

File → Open Folder ...

6. Kjør følgende kommando i VS Code:

`Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only`

Sjekkliste ved tekniske problemer

Hvis prosjektet du oppretter med den utdelte koden ikke kompilerer (før du har lagt til egne endringer) bør du rekke opp hånden og be om hjelp. Mens du venter kan du prøve følgende:

1. Sjekk at prosjektmappen er lagret i mappen C:\temp.
2. Sjekk at navnet på mappen **IKKE** inneholder norske bokstaver (*Æ, Ø, Å*) eller mellomrom. Dette kan skape problemer når du prøver å kjøre koden i VS Code.
3. Sørg for at du er inne i riktig mappe i VS Code.
4. Kjør følgende TDT4102-kommando:

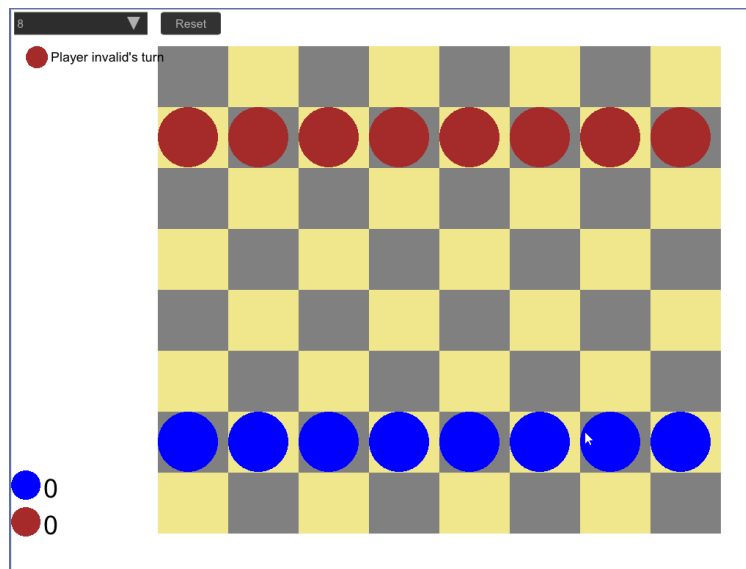
`Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only`

5. Prøv å kjøre koden igjen (`Ctrl+F5` eller `Fn+F5`).

6. Hvis det fortsatt ikke fungerer, lukk VS Code vinduet og åpne det igjen. Gjenta deretter steg 5-6.

Introduksjon

Bondesjakk er en forenklet form for sjakk. Reglene er for det meste de samme som i sjakk, men man har kun én type sjakkbrikke: bonden. I tillegg handler det ikke lenger om å sette kongen i sjakk, men om å komme seg over til den andre siden av brettet med en av bøndene sine. Den som først kommer seg over til den andre siden vinner spillet. I den utdelte koden har du et spill med veldig enkel grafikk. Dessverre mangler det noen biter av grensesnittet. Oppgaven din er å implementere de delene av spillet som mangler og forbedre noen andre aspekter av spillet.

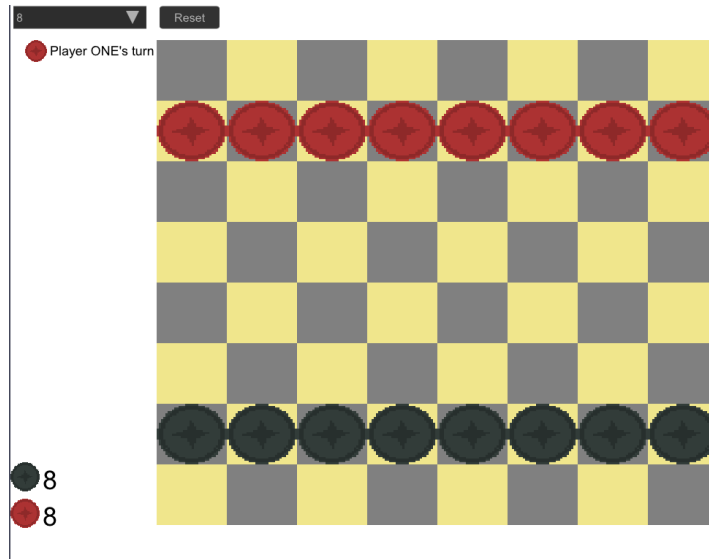


Figur 1: Skjerm bilde av kjøring av den utdelte koden.

Den utdelte koden inneholder mange filer, men du skal kun forholde deg til `Tasks.cpp`, `ImageAtlas.cpp`, `BoardGame.cpp`, og `Board.cpp` og tilhørende header-filer. De andre filene trenger du ikke se på for å kunne besvare oppgavene. All informasjon som trengs for å besvare oppgavene står oppgitt i oppgaveteksten. Før du starter må du sjekke at den umodifiserte utdelte koden kjører uten problemer. Du skal se det samme vinduet som i Figur 1.

Slik fungerer applikasjonen

Fra start har du et spill med enkelt utseende og der deler av grensesnittet. Når applikasjonen er ferdig implementert (figur 2) er spillet fungerende med oppdatert grafikk og fungerende grensesnitt.



Figur 2: Skjerm bilde av den ferdige applikasjonen.

Hvordan besvare oppgavene

Hver oppgave i del 3 har en unik kode for å gjøre det lettere for deg å vite hvor du skal fylle inn svarene dine. Koden er på formatet <T><siffer> (TS), der sifrene er mellom 1 og 11 (T1 - T11). For hver oppgave vil man finne to kommentarer som skal definere henholdsvis begynnelsen og slutten av svaret ditt. Kommentarene er på formatet: // BEGIN: TS og // END: TS.

For eksempel kan en oppgave se slik ut:

```
// TASK T1
bool foo(int x, int y) {
// BEGIN: T1
// Write your answer to assignment T1 here, between the //BEGIN: T1
// and // END: T1 comments. You should remove any code that is
// already there and replace it with your own.

return false;

// END: T1
}
```

Etter at du har implementert løsningen din bør du ende opp med følgende:

```
// TASK T1
bool foo(int x, int y) {
// BEGIN: T1
// Write your answer to assignment T1 here, between the //BEGIN: T1
// and // END: T1 comments. You should remove any code that is
// already there and replace it with your own.
    return (x + y) % 2 == 0;
// END: T1
}
```

Det er veldig viktig at alle svarene dine føres mellom slike par av kommentarer. Dette er for å støtte sensurmekanikken vår. Hvis det allerede er skrevet kode *mellom* BEGIN- og END-kommentarene til en oppgave i det utdelte kodeskjelettet, så kan du, og ofte bør du, erstatte denne koden med din egen implementasjon. All kode som står *utenfor* BEGIN- og END-kommentarene **SKAL** du la stå som den er. I oppgave T8 er det ikke noen funksjonsdeklarasjon eller funksjonsdefinisjon mellom BEGIN- og END-kommentarene, så her må du skrive all kode selv, og det er viktig at du skriver all koden din mellom BEGIN- og END-kommentarene for å få uttelling for oppgaven.

Merk: Du skal **IKKE** fjerne BEGIN- og END-kommentarene.

Hvis du synes noen av oppgavene er uklare kan du oppgi hvordan du tolker dem og de antagelsene du gjør for å løse oppgaven som kommentarer i koden du leverer.

Tips: Trykker man CTRL+SHIFT+F og søker på BEGIN: får man snarveier til starten av alle oppgavene listet opp i utforskervinduet slik at man enkelt kan hoppe mellom oppgavene. For å komme tilbake til det vanlige utforskervinduet kan man trykke CTRL+SHIFT+E.

Oppgavene

Få orden på brettet (30 poeng)

```
enum class Player {
    NONE, ONE, TWO
};

struct Tile {
    Player player;
    bool firstMove{false};
};

struct Board {

    // Constructors...

    static Board create_blank(int size_);

    //! Get the size n of the board (n x n)
    int get_size() const;

    //! Move a piece from `from` to `to`
    void move(TDT4102::Point from, TDT4102::Point to);

    //! Get a reference to the cell at (x, y)
    Tile &cell_at(int x, int y);

    //! Get a reference to the cell at (x, y)
    const Tile &cell_at(int x, int y) const;

    //! Get a constant reference to the tile storage
    std::vector<Tile> const &get_cells() const;

private:
    int size;
    std::vector<Tile> cells;
};
```

Figur 3: Klassedeklarasjoner for Player, Tile og Board

```

struct Entity {
    TDT4102::Point get_position();
    void set_position(TDT4102::Point new_pos);

    int get_width();
    void set_width(int width);

    int get_height();
    void set_height(int height);

    // ...
}

struct BoardGame : public Entity {
    BoardGame(int size);

    // One of the tasks
    BoardGame(const Board &board_);

    // One of the tasks
    bool inside_interaction_zone(TDT4102::Point pt);

    //! Resets and resizes the highlighted vector
    void resize(int size);

    //! The currently selected tile.
    TDT4102::Point selected;

private:
    // A vector that marks highlighted tiles with true and others with false.
    std::vector<bool> highlighted;

    // The board in use by the game.
    Board board;
};

```

Figur 4: Klassedeklarasjoner for BoardGame og Entity. Merk at BoardGame arver fra Entity.

1. (5 points) **T1: Hvor stort er brettet?**

Board-klassen (Figur 3) har en medlemsvariabel `size` som sier hvor stort brettet er i én dimensjon. Medlemsfunksjonen `get_size` blir brukt i koden til å hente ut verdien til `size`, men for øyeblikket returnerer `get_size` alltid 8, så brettet har alltid størrelsen 8×8 . Endre funksjonen `get_size` så den returnerer verdien til `size`.

Implementer funksjonen `Board::get_size` i `Board.cpp`

2. (10 points) **T2: Hvilken spiller?**

En spiller blir representert av en enum-klasse kalt `Player` (Figur 3). Vi ønsker nå en funksjon som kan ta en instans av `Player` som parameter og returnere en streng som representerer navnet til spilleren. Hvis spilleren er `Player::ONE` skal "ONE" returneres, mens hvis spilleren er `Player::TWO` skal "TWO" returneres. Utenom dette skal funksjonen returnere "invalid".

Implementer funksjonen `player_key` i `Tasks.cpp`.

- Hvis spilleren er `Player::ONE`, returner "ONE".
- Hvis spilleren er `Player::TWO`, returner "TWO".
- Hvis spilleren hverken er `Player::ONE` eller `Player::TWO`, returner "invalid".

Når oppgave T2 er gjort skal du kunne se hvilken spiller sin tur det er øverst til venstre på skjermen.

3. (15 points) T3: Innenfor brettet?

For å kunne interagere med spillet må det være mulig å klikke på brettet. Funksjonaliteten for å klikke er stort sett allerede implementert, men én av funksjonene må endres. Funksjonen `inside_interaction_zone` i `BoardGame.cpp` skal returnere `true` dersom et punkt ligger innenfor selve brettet, men for øyeblikket returnerer den alltid `true`.

Funksjonen `inside_interaction_zone` tar inn et punkt, `point`, som parameter, og skal avgjøre hvorvidt `point` ligger på innsiden av brettet. For å hente ut informasjon om brettet kan man bruke funksjonen `get_position()` som returnerer et `TDT4102::Point` som representerer posisjonen til det øverste venstre hjørnet til brettet. Funksjonene `get_width()` og `get_height()` kan brukes for å hente ut henholdsvis bredden og høyden til brettet. Du kan se deler av definisjonen til `BoardGame` i Figur 4, og hele definisjonen finnes i `BoardGame.h`.

Endre funksjonen `BoardGame::inside_interaction_zone` i `BoardGame.cpp`.

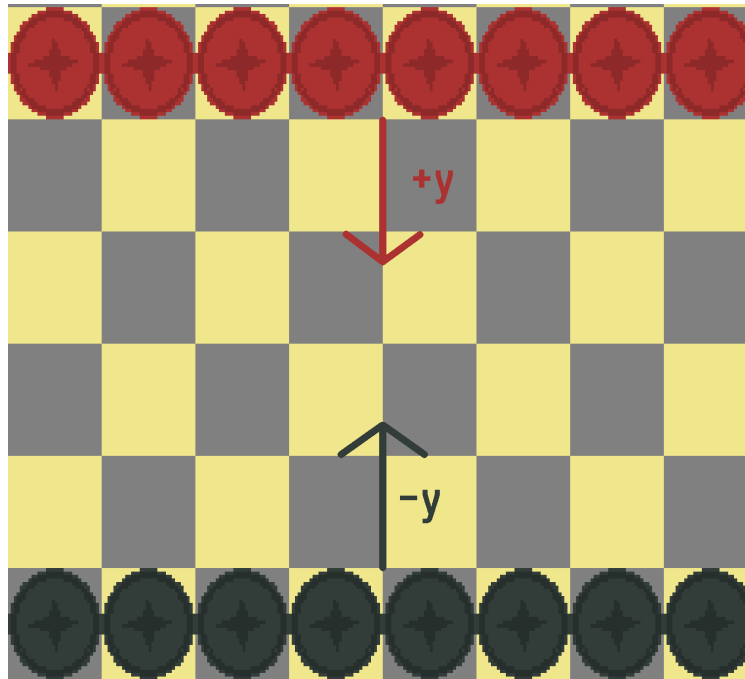
Gitt `point`, returner `true` dersom:

- `point.x` ligger mellom b_x og $b_x + w$, og
- `point.y` ligger mellom b_y og $b_y + h$,

der b_x og b_y er henholdsvis x - og y -koordinaten til det øvre venstre hjørnet til brettet, og w og h er henholdsvis bredden og høyden til brettet.

Når oppgave T3 er gjort vil det ikke skje noe på skjermen dersom man klikker utenfor selve brettet.

Visuelle detaljer og regler (70 poeng)



Figur 5: Illustrasjon av bevegelsesretning for bøndene til spiller én (øverst) og spiller to (nederst). $+y$ betyr en positiv endring i y , mens $-y$ betyr en negativ endring i y .

4. (15 points) T4: Marsjerer vi i riktig retning?

Dette skal være bondesjakk, men slik spillet er nå kan vi flytte bøndene i enhver retning, altså både opp, ned, til venstre og til høyre. Som i tradisjonell sjakk, kan bøndene kun bli flyttet i én retning, opp eller ned avhengig av spilleren. Spilleren som starter øverst på brettet skal bevege seg nedover, mens spilleren som starter nederst på brettet skal bevege seg oppover. Riktig retning for hver spiller er vist i Figur 5.

Vi skal nå implementere funksjonen `is_valid_direction` i `Tasks.cpp`, som basert på parameterne `from`, `to` og `player` skal returnere en boolsk verdi som sier om et trekk fra `from` til `to` fra spilleren `player` er i riktig retning. For spiller én (`Player::ONE`) er et trekk i riktig retning et trekk som gir positiv endring i y -koordinaten, mens det for spiller to (`Player::TWO`) er et trekk som gir negativ endring i y -koordinaten.

Implementer funksjonen `is_valid_direction` i `Tasks.cpp`

- Returner `true` hvis differansen mellom y -koordinatene til punktet `from` og `to` er positiv og `player` er `Player::ONE`.
- Returner `true` hvis differansen mellom y -koordinatene til punktet `from` og `to` er negativ og `player` er `Player::TWO`.
- Returner `false` ellers.

5. (15 points) T5: Bilder til brikker

Brikkene blir nå vist på skjermen som ensfargede røde og blå sirkler, noe som er litt kjedelig. Vi vil derfor gjøre brikkene litt finere å spille med ved å laste inn bilder for brikkene. For å hjelpe med dette finnes klassen `ImageAtlas`.

Klassen `ImageAtlas` har en medlemsfunksjon `get_image` definert i `ImageAtlas.cpp` som brukes til å hente ut bilder, men implementasjonen til denne funksjonen er ufullstendig. Slik implementasjonen er nå returnerer funksjonen en standard `std::shared_ptr`, men vi vil at funksjonen skal returnere en `std::shared_ptr` til et `TDT4102::Image`.

I `ImageAtlas::get_image` finnes allerede variabelen `container`, som er et `std::unordered_map` med nøkkeltipe `std::string` og element-type `std::shared_ptr<TDT4102::Image>`. Vi vil bruke parameteren `key` for å hente ut en `std::shared_ptr` til et `TDT4102::Image`. Dersom `container` ikke inneholder et element for nøkkelen `key`, vil vi at funksjonen skal returnere en standard `std::shared_ptr`.

Fullfør implementasjonen av `ImageAtlas::get_image` i `ImageAtlas.cpp`.

- Bruk parameteren `key` til å hente ut og returnere en `std::shared_ptr` til et `TDT4102::Image` fra `container`.
- Returner en standard `std::shared_ptr` hvis `container` **ikke** inneholder et element for nøkkelen `key`.

Når oppgave T5 er gjort skal du kunne se at brikkene på skjermen har gått fra å se ut som i Figur 1 til å se ut som i Figur 2.

6. (20 points) T6: Tell brikkene

Til venstre for brettet finnes det to tellere som skal vise hvor mange brikker hver spiller har på brettet. Disse tellerne er avhengige av funksjonen `count_chips` i `Tasks.cpp` som enda ikke er implementert, og de viser derfor at spillerne har 0 brikker hver på brettet uansett hvordan brettet ser ut.

Funksjonen `count_chips` tar en vektor av `Tile`-objekt som argument (Figur 3) og skal returnere antall ruter på brettet som er okkupert av `Player::ONE` og antall ruter på brettet som er okkupert av `Player::TWO`.

Implementer funksjonen `count_chips` i `Tasks.cpp`.

- Iterer gjennom vektoren `cells` av `Tile`-objekt og tell antall ruter på brettet som er okkupert av spilleren `Player::ONE` og antall ruter på brettet som er okkupert av spilleren `Player::TWO`.
- Returner antall ruter på brettet som er okkupert av hver spiller.

Hint: Det finnes allerede et `std::pair<int, int>` kalt `counts` i funksjonen som du kan bruke til å telle rutene som okkuperes av hver spiller. Et `std::pair` har to felt kalt `first` og `second` som du kan lese fra og skrive til ved å bruke notasjonen `counts.first` og `counts.second`.

Når oppgave T6 er gjort skal du kunne se at antall ruter på brettet som er okkupert av spillerne vises i tellerne til venstre for brettet på skjermen.

7. (20 points) T7: Vis gode trekk.

Når man klikker på en rute med en brikke på brettet, blir ruten markert med en annen farge enn den som var der fra før. Vi ønsker i tillegg å vise hvor brikken på den valgte ruten kan flyttes til. For å oppnå dette skal vi nå implementere funksjonen `BoardGame::highlight` i `BoardGame.cpp`.

For å gjøre denne oppgaven må du bruke funksjonen `Rules::can_move` som er definert i `Rules.cpp`. Du trenger ikke å bry deg om implementasjonsdetaljene, men du kan se funksjonsdeklarasjonen i Figur 7. Denne funksjonen returnerer en boolsk verdi basert på om et trekk fra ruten med koordinat `from` til ruten med koordinat `to` er gyldig. Den tar inn en referanse til brettet, koordinaten til ruten trekket starter i (`from`), koordinaten til ruten trekket slutter i (`to`) og et `Player`-objekt som sier hvem sin tur det er (`turn`.)

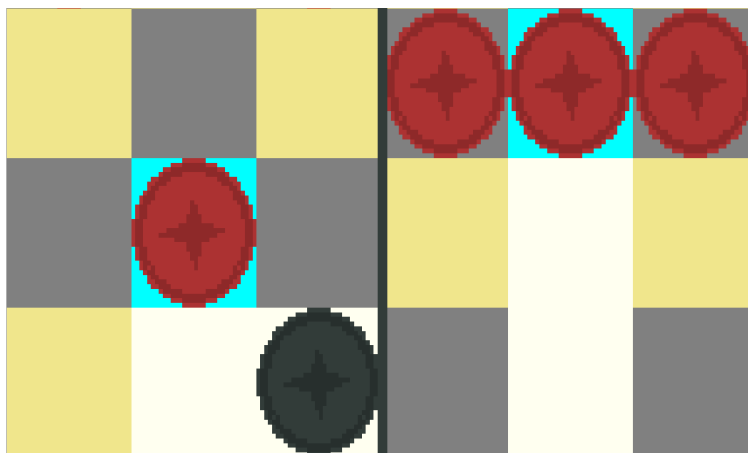
For å kunne markere alle rutene som en spiller kan flytte en gitt brikke til må vi sjekke alle rutene på brettet. For hver rute på brettet skal vi bruke funksjonen `can_move` for å avgjøre om et trekk fra `selected` til ruten er gyldig og merke ruten med verdien som `can_move` returnerer.

Implementer `BoardGame::highlight` i `BoardGame.cpp`.

For hver rute (x, y) på brettet:

- Sett `highlighted[y * size + x] = can_move(board, from, to, turn)`
- Gi riktig argument til `can_move`-funksjonen:
 - Parameteren `board` skal motta en instans av `Board`.
 - Parameteren `from` skal motta koordinaten til ruten med brikken man ønsker å flytte.
 - Parameteren `to` skal motta koordinaten til ruten man sjekker om man kan flytte til.
 - Parameteren `turn` skal motta enum-verdien for spilleren hvis tur det er.

Når oppgave T7 er gjort skal du kunne se gyldige trekk markert på brettet slik som i Figur 6 når du klikker på en rute med en brikke.



Figur 6: Skjermbilde av applikasjonen etter `BoardGame::highlight` er implementert. Bildet viser to situasjoner. *Venstre:* en brikke som kan flytte rett frem eller eliminere brikken nedenfor til høyre. *Høyre:* en brikke som ikke har blitt flyttet før og kan flytte én rute eller to ruter nedover.

```

struct Rules {
    static bool can_move(Board &board, TDT4102::Point from, TDT4102::Point to,
        Player turn);

    static bool move(Board &board, TDT4102::Point from, TDT4102::Point to,
        Player turn);

    static Player winner(Board &board);
};

```

Figur 7: Klassedeklarasjon for Rules

Kopiering, overlasting og aksess (80 poeng)

8. (15 points) T8: Likhetsoperator for Point

Strukturen `TDT4102::Point` brukes mye i denne applikasjonen. Strukturen kan du se under, selv om du kanskje allerede er godt kjent med den:

```

struct TDT4102::Point {
    int x;
    int y;
};

```

Vi ønsker å kunne sjekke om to `TDT4102::Point` er like, og for å kunne gjøre det må vi overlaste `==`-operatoren. To punkter `p1` og `p2` er like dersom `p1.x == p2.x` og `p1.y == p2.y`.

Overlast likhetsoperatoren (`operator==`) for `TDT4102::Point` i `Tasks.cpp`.

- `operator==` skal returnere `true` hvis punktene er like.
- `operator==` skal returnere `false` hvis punktene er ulike.

Merk: Oppgaven ber deg om å skrive hele overlastingen. Husk å skrive svaret ditt innenfor `BEGIN-` og `END-`kommentarene til oppgave T8 for å få uttelling for oppgaven.

Når oppgave T8 er gjort vil du kunne se at det står "Point equality overload in use!" når du starter applikasjonen.

9. (15 points) T9: Callback funksjon for Reset knappen

Vi ønsker å tilby spillerne muligheten til å tilbakestille spillet, og kanskje til og med velge større brett å spille på. For øyeblikket skjer det ingenting dersom man trykker på Reset-knappen fordi funksjonen som kalles når Reset-knappen blir trykket på, `reset.button_callback`, ikke er implementert. Dette skal vi fikse nå.

Det er noen detaljer som er viktige å ha i bakhodet når du jobber med denne oppgaven. Det finnes to globale unike pekere å forholde seg til. `list` er en `std::unique_ptr` til et `TDT4102::DropDownList`-objekt, og representerer nedtrekksmenyen til venstre for Reset-knappen. `game_ptr` er en `std::unique_ptr` til et `BoardGame`-objekt, og representerer brettet man ser på skjermen. Definisjonene til de globale unike pekerne finnes i `main.cpp`.

Når man trykker på Reset-knappen skal objektet som `game_ptr` peker til skiftes ut med en ny instans av `BoardGame`. Vi kan derimot ikke bare lage en ny instans av `BoardGame`, siden posisjonen, bredden og høyden da ikke nødvendigvis blir de samme som før. Det første vi skal gjøre er derfor å hente ut og lagre den nåværende posisjonen, bredden og høyden så vi kan oppdatere verdiene til den nye `BoardGame`-instansen med de samme verdiene som den gamle `BoardGame`-instansen hadde. Størrelsen til den nye `BoardGame`-instansen skal være gitt av verdien i nedtrekksmenyen til venstre for Reset-knappen.

Implementer `reset_button_callback` i `Tasks.cpp`.

Funksjonen skal:

- Lagre posisjonen, bredden og høyden til `BoardGame`-instansen den globale variabelen `game_ptr` peker til.
- Hente ut verdien til `TDT4102::DropDownList`-instansen som den globale variabelen `list` peker til.
- Konvertere strengen hentet ut i det forrige steg til et heltall.
- Overskrive `game_ptr` sin `BoardGame`-instans med en ny `BoardGame`-instans som har størrelse lik verdien som ble hentet ut og konvertert i de to forrige stegene.
- Sette bredden, høyden og posisjonen til den nye `BoardGame`-instansen lik verdiene som ble hentet ut i første steg.

Hint 1: Du kan bruke konstruktøren til `BoardGame` med en heltallsparameter for å lage et helt nytt `BoardGame`-objekt med alle brikkene satt på plass.

Hint 2: Medlemsfunksjonene til `BoardGame` er arvet fra `Entity` (Figur 4).

Når oppgave T9 er gjort skal man kunne bruke Reset-knappen til å tilbakestille spillet. Det nye brettet skal ha samme posisjon, bredde og høyde som det forrige brettet, og størrelse gitt av nedtrekksmenyen til venstre for Reset-knappen.

10. (20 points) **T10: Kopikonstruktør for BoardGame**

Nå skal vi implementere en kopikonstruktør for klassen `BoardGame` som tar utgangspunkt i et `Board` objekt. Målet med oppgaven er å sikre at kopikonstruktøren initialiserer et nytt `BoardGame`-objekt som tar i bruk brettet (`Board`) den mottar.

Implementer kopikonstruktøren til `BoardGame` i `BoardGame.cpp`

Kopikonstruktøren skal sørge for at:

- Det blir spiller én (`Player::ONE`) sin tur (sett verdien til `turn`).
- Vinneren tilbakestilles til `Player::NONE` (sett verdien til `winner`).
- Spillet settes i gang ved å sette `enabled` til `true`.
- `highlighted` vektoren tømmes.
- `highlighted` vektoren sin størrelse tilpasses størrelsen til `Board`-objektet den mottar.

11. (30 points) **T11: Lese tilstand fra fil**

Vi ønsker nå å gjøre det mulig å gjenoppta et uferdig spill. For å gjøre det må vi både kunne lagre tilstanden til et spill i en fil, og laste inn et spill fra en fil. Funksjonen for å lagre tilstanden til et spill er ikke implementert, men det finnes et "uferdig" spill lagret som tekst i filen `state`.

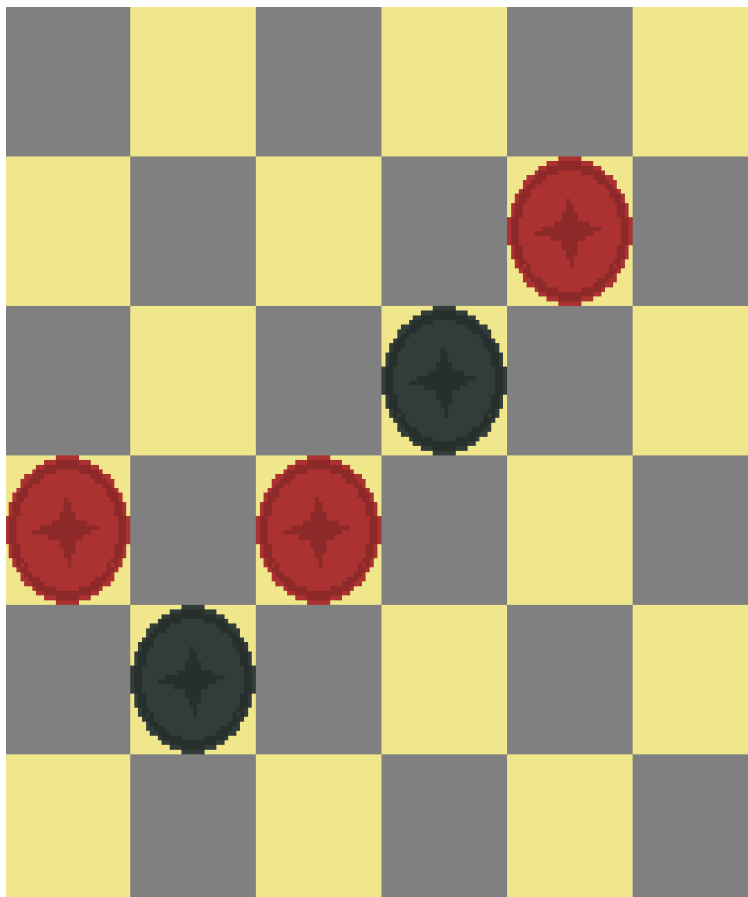
Du skal implementere funksjonen `load_board` i `Board.cpp` som leser inn en spilltilstand fra en fil og oppretter et `Board`-objekt som representerer tilstanden beskrevet i filen. Filen inneholder informasjon om størrelsen til brettet og tilstanden til hver rute. Første linje i filen er et heltall N som representerer størrelsen til brettet i hver dimensjon ($N \times N$). De neste N linjene er N heltall separert av mellomrom som representerer tilstanden til hver rute på brettet. En rute kan være tom (0), inneholde en brikke fra spiller én (1), eller inneholde en brikke fra spiller to (2). Et eksempel på hvordan dette ser ut kan sees i filen `state` eller i Figur 9.

Implementer funksjonen `load_board` i `Board.cpp`.

- Åpne filen som er gitt av parameteren `path`.
- Kontroller at filen ble åpnet og kast en `std::runtime_error` med en passende feilmelding hvis den ikke ble det.
- Les den første linjen i filen for å hente ut størrelsen til brettet og initialiser et `Board`-objekt med denne størrelsen.
- For hver av de neste N linjene:
 - Les linjen.
 - Konverter hvert heltall til den tilsvarende enum-verdien:
 - $0 \rightarrow \text{Player}::\text{NONE}$
 - $1 \rightarrow \text{Player}::\text{ONE}$
 - $2 \rightarrow \text{Player}::\text{TWO}$
 - Oppdater `Board`-objektet med tilstanden til rutene.
- Lukk filen når all data er lest og prosessert.
- Returner det nye `Board`-objektet.

Hint: Bruk `cell.at` funksjonen i `Board`-klassen for å gjøre det lettere å finne riktig rute. `cell.at` tar inn to heltall x og y , og returnerer en referanse til ruten (`Tile &`) med koordinaten (x,y) .

Når oppgave T11 er gjort skal du se det samme brettet som i Figur 8 på skjermen.



Figur 8: Korrekt innlastet fil for oppgave T11.

```

6
0 0 0 0 0 0
0 0 0 0 1 0
0 0 0 2 0 0
1 0 1 0 0 0
0 2 0 0 0 0
0 0 0 0 0 0

```

Figur 9: Eksempelfil for et Brett med størrelse 6×6 .

i Oppgave/utdelt kode del 3

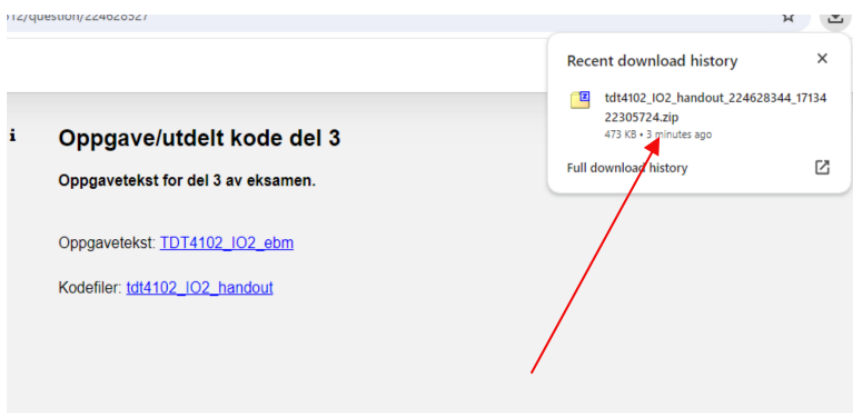
Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102_IO2_ebm](#)

Kodefiler: [tdt4102_IO2_handout](#)



(a) Steg 1




i Oppgave/utdelt kode del 3

Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102_IO2_ebm](#)

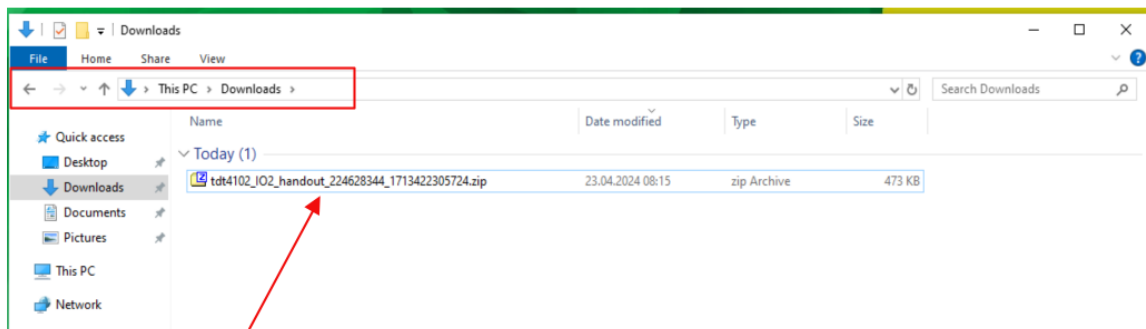
Kodefiler: [tdt4102_IO2_handout](#)

Recent download history

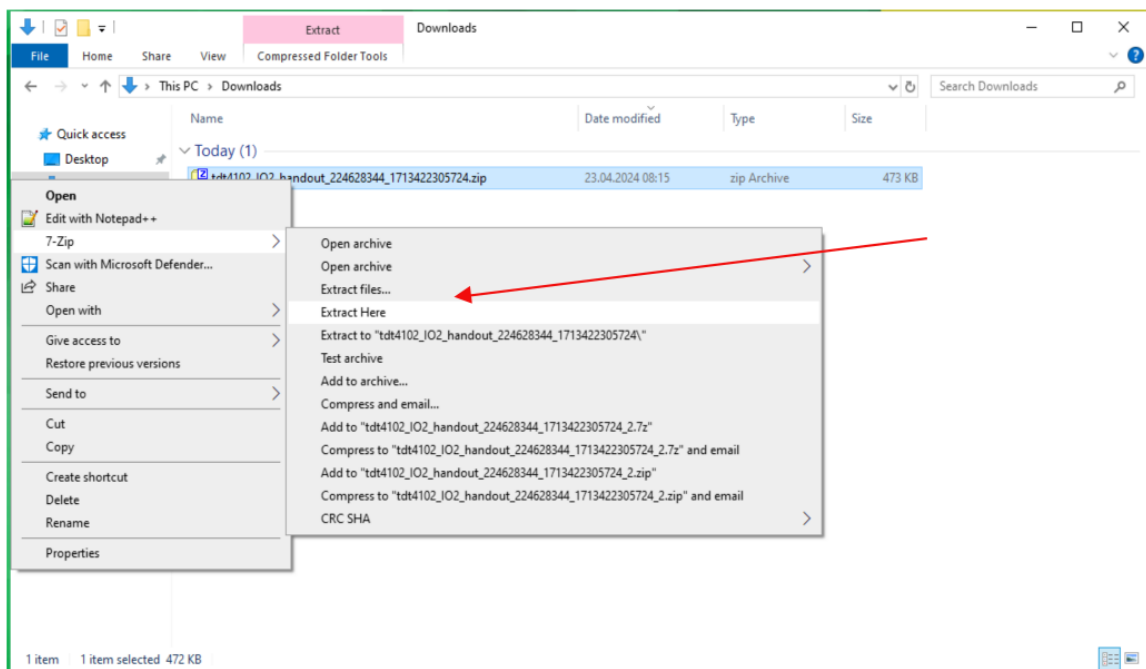
 tdt4102_IO2_handout_224628344_17134_22305724.zip

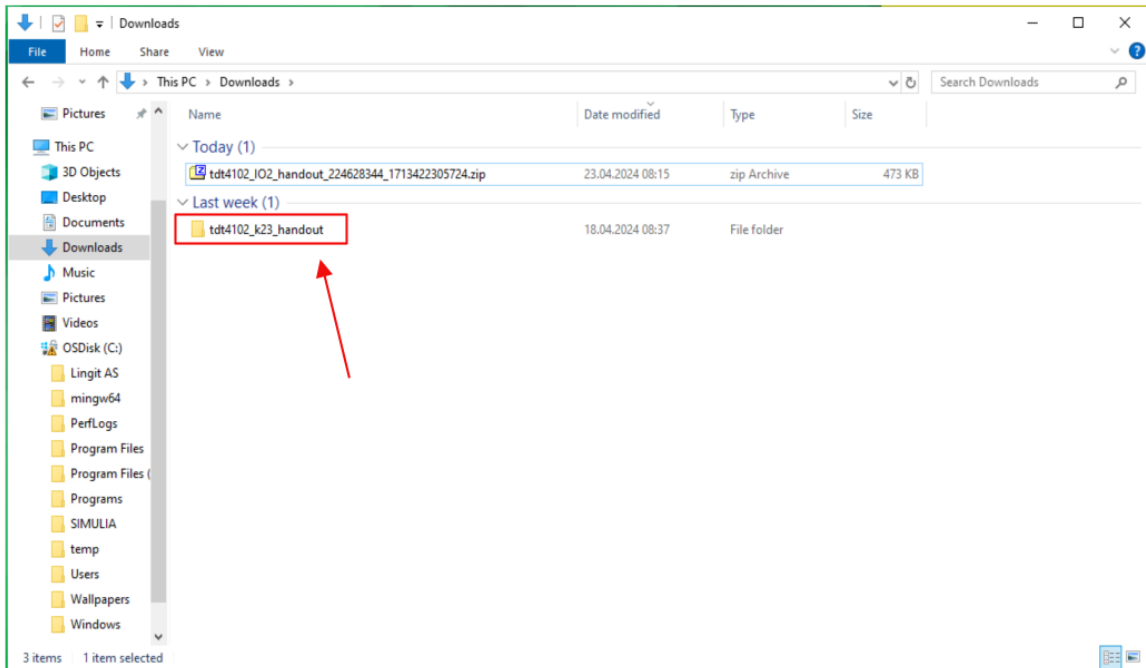
473 KB • 3 minutes ago

Full download history

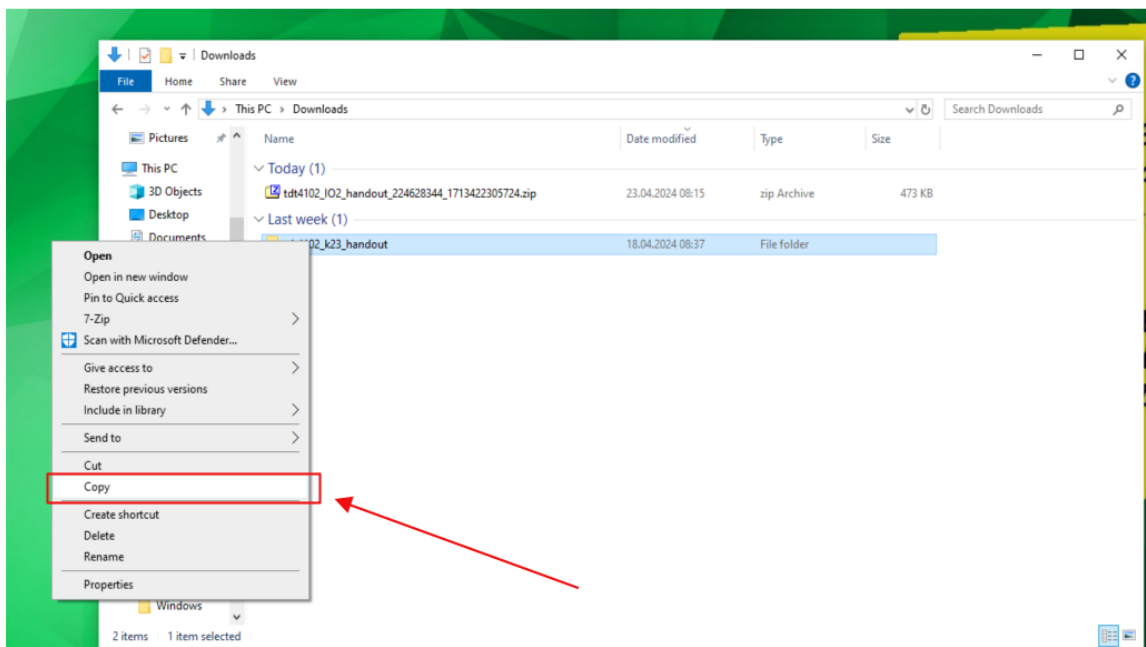


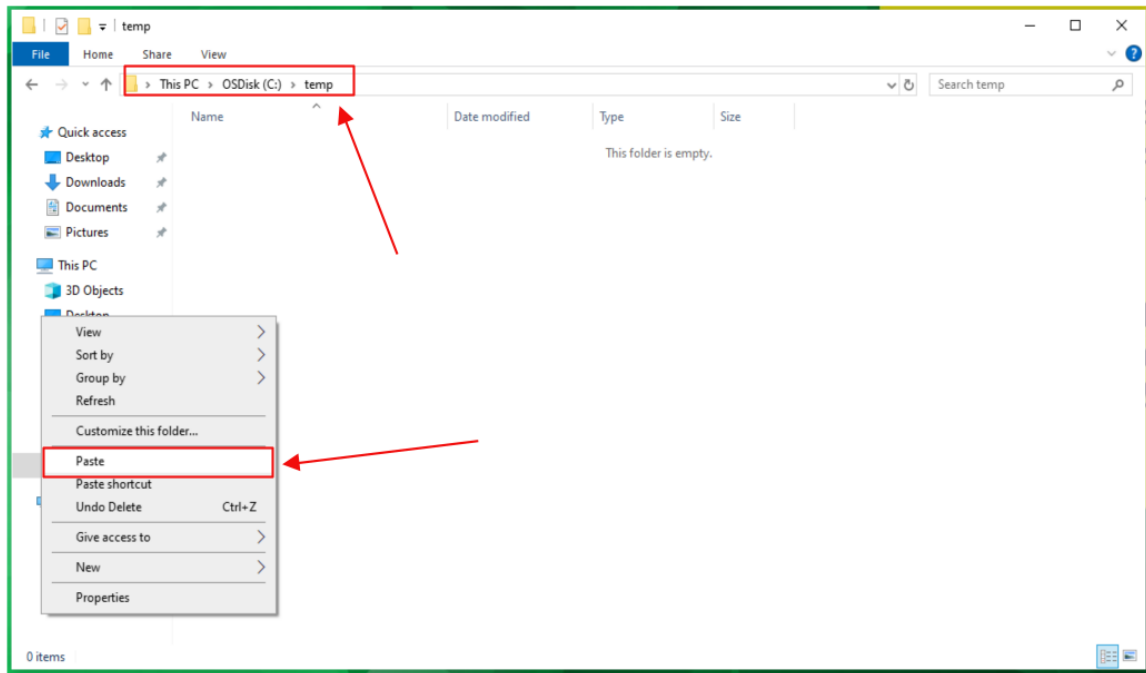
(c) Steg 2



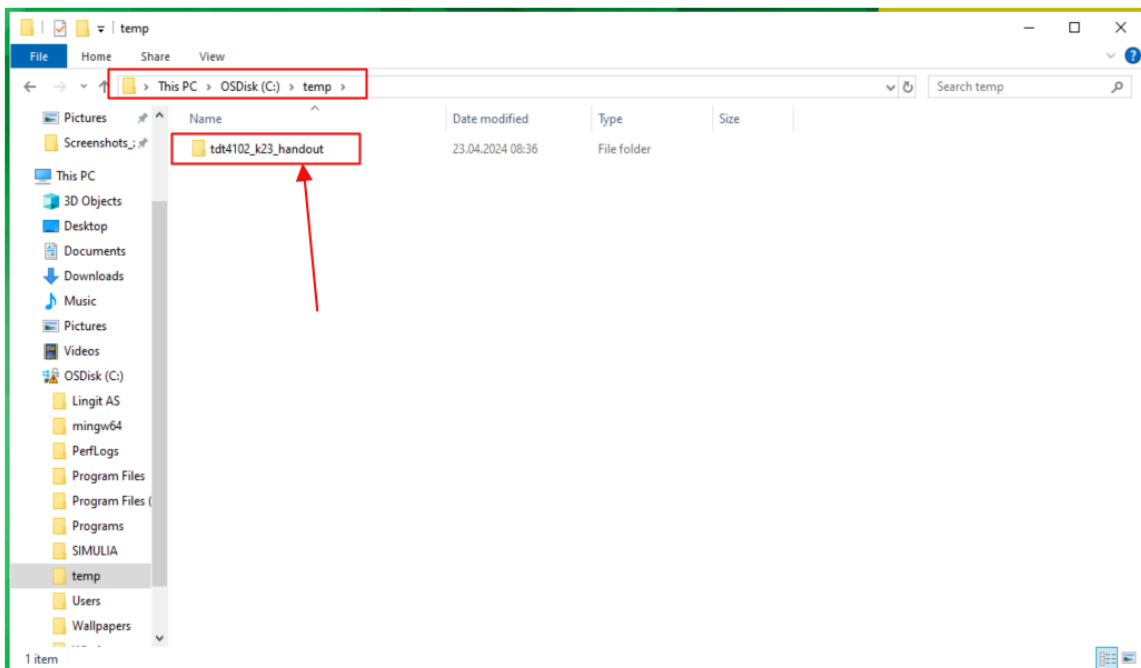


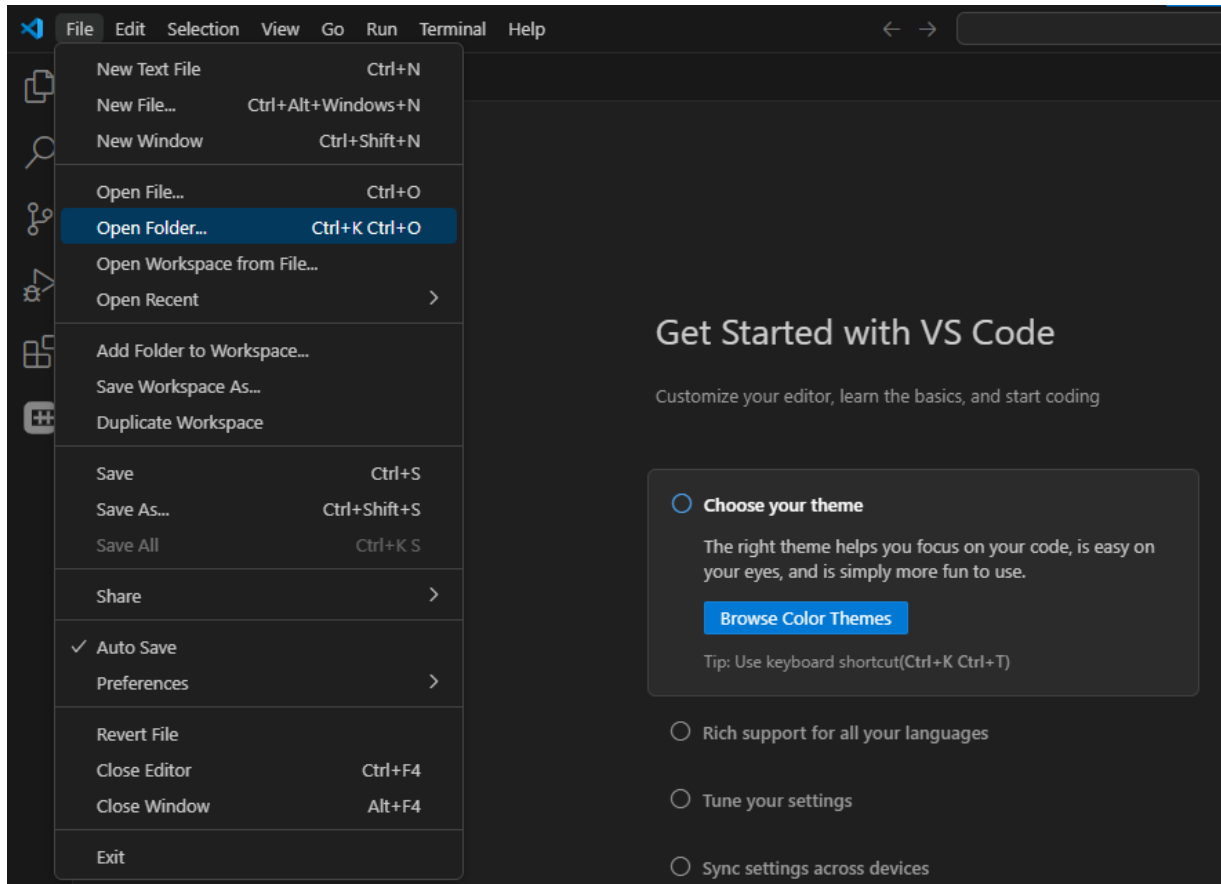
(e) Steg 3



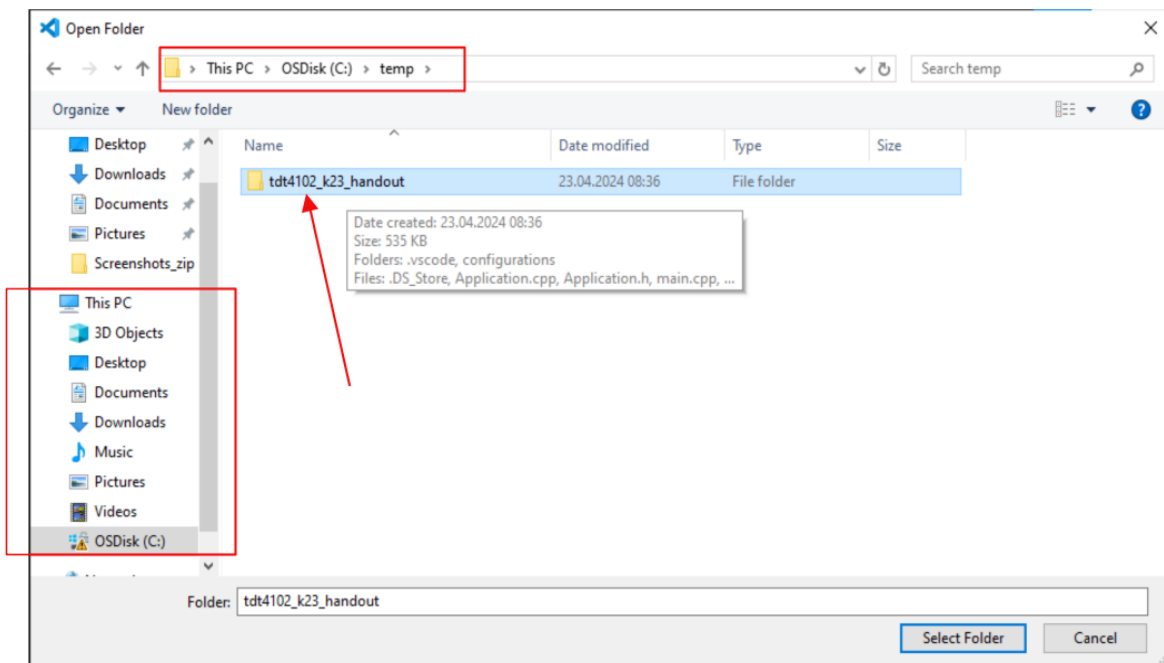


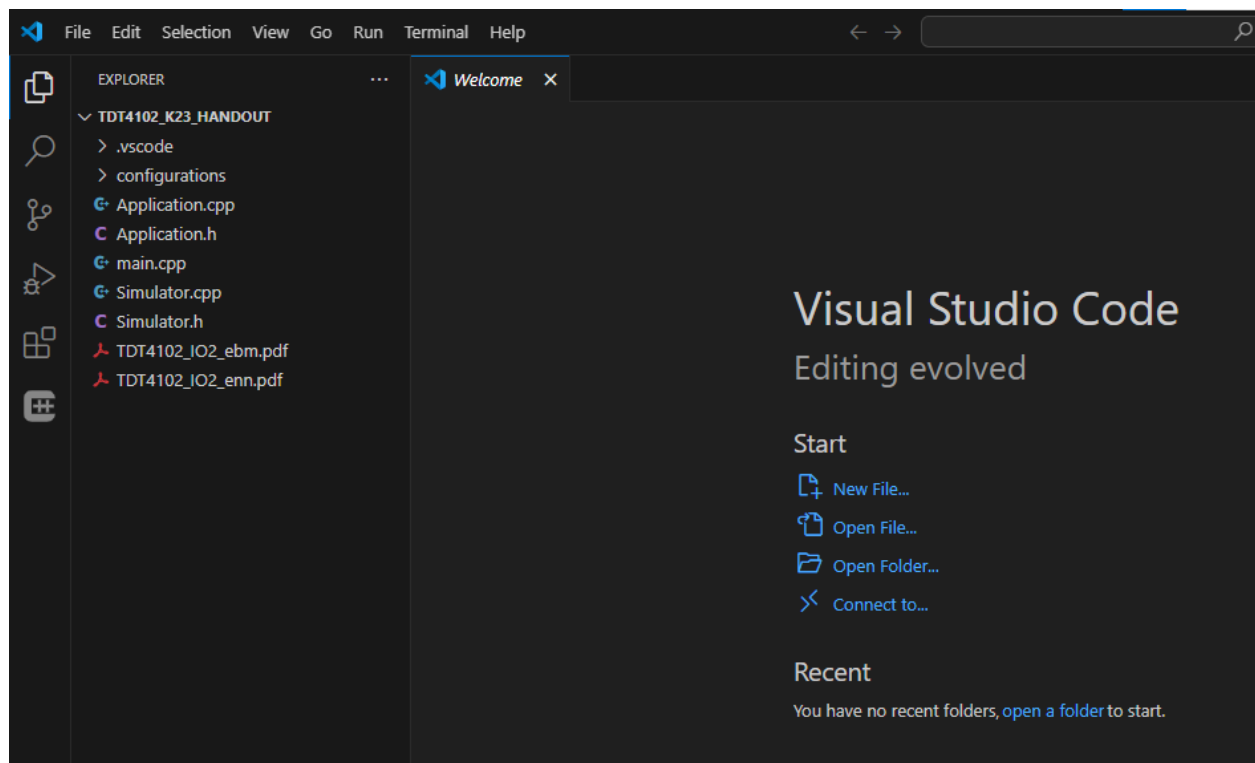
(g) Steg 4





(i) Steg 5





(k) Steg 5

Figur 10: Nedlasting og oppsett av den utdelte koden.