

## Del 3: Bondesjakk

Del 3 av eksamen er programmeringsoppgåver. Denne delen inneheld **11 oppgåver** som til saman gir ein maksimal poengsum på 180 poeng og tel **ca. 60 %** av eksamen. Kvar oppgåve gir maksimal poengsum på **5 - 30 poeng** avhengig av vanskegrad og arbeidsmengd.

Den utdelte koden inneheld kompillerbare (og kjørbare) .cpp- og .h-filer med forhåndskoda delar og ei full beskrivelse av oppgåvene i del 3 som ei PDF. Etter å ha lasta ned koden står du fritt til å bruke eit utveklingsmiljø (VS Code) for å jobbe med oppgåvene.

Du kan lasta ned .zip-fila med den utdelte koden frå Inspira. I neste seksjon finn du ei beskriving av korleis du gjer dette. Før du leverer eksamen, må du hugse å lasta opp ei .zip-fil med den utdelte koden og endringene du har gjort når du har løyst oppgåvene.

## Nedlasting og oppsett av den utdelte koden

Dei seks stega under beskriv korleis du kan lasta ned og setje opp den utdelte koden. Ei bildebeskriving av stega 1 til 5 kan du se i Figur 10.

1. Last ned .zip-fila frå Inspira
2. Lokaliser fila i File Explorer (Downloads-mappa)
3. Pakk ut .zip-fila ved å høgreklikke på fila og vel:  
*7-Zip → Extract here*
4. Kopier mappa som dukkar opp til C:/Temp
5. Opne mappa som ligg i C:/Temp i VS Code ved å velje:  
*File → Open Folder ...*

6. Køyr følgjande kommando i VS Code:

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

## Sjekkliste ved tekniske problem

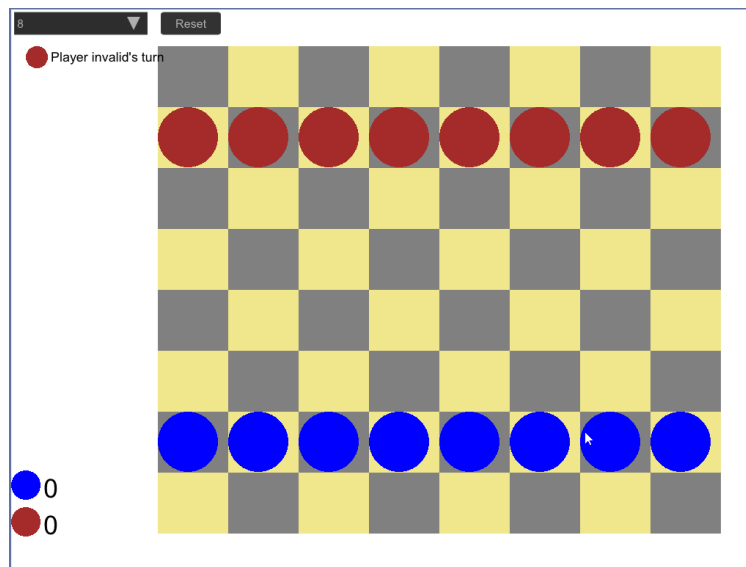
Viss prosjektet du opprettar med den utdelte koden ikkje kompilerer (før du har lagt til eigne endringer) bør du rekkje opp hânda og be om hjelp. Medan du ventar kan du prøva følgjande:

1. Sjekk at prosjektmappa er lagret i mappa C:\temp.
2. Sjekk at namnet på mappa **IKKJE** inneheld norske bokstavar (*Æ, Ø, Å*) eller mellomrom. Dette kan skapa problem når du prøver å køyre koden i VS Code.
3. Sørg for at du er inne i rett mappe i VS Code.
4. Køyr følgjande TDT4102-kommando:  
(a) `Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only`
5. Prøv å køyra koden igjen (`Ctrl+F5` eller `Fn+F5`).

6. Viss det framleis ikkje fungerer, lukk VS Code vindauget og opne det igjen. Gjenta deretter steg 5-6.

## Introduksjon

Bondesjakk er ei forenkla form for sjakk. Reglane er for det meste dei same som i sjakk, men ein har kun éin type sjakkbrikke: bonden. I tillegg handlar det ikkje lenger om å setje kongen i sjakk, men om å kome seg over til den andre sida av brettet med ein av bøndene sine. Den som først kjem seg over til den andre sida vinn spelet. I den utdelte koden har du eit spel med veldig enkel grafikk. Dessverre manglar det nokon bitar av grensesnittet. Oppgåva din er å implementere dei delane av spelet som manglar og forbetre nokon andre aspekt av spelet.

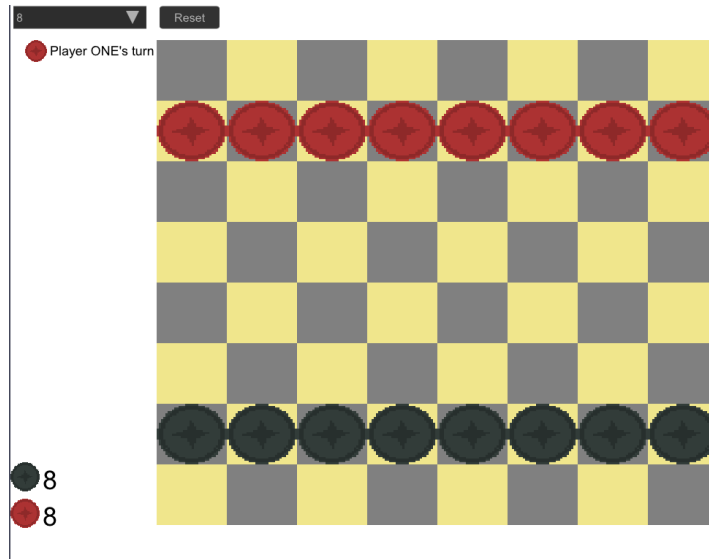


**Figur 1:** Skjerm bilde av køyring av den utdelte koden.

Den utdelte koden inneheld mange filer, men du skal berre forhalde deg til `Tasks.cpp`, `ImageAtlas.cpp`, `BoardGame.cpp` og `Board.cpp` og tilhøyrande header-filer. Dei andre filene treng du ikkje sjå på for å kunne svare på oppgåvene. All informasjon som trengst for å svare på oppgåvene står oppgjeve i oppgåveteksten. Før du startar må du sjekka at den (umodifiserte) utdelte koden køyrer utan problem. Du skal sjå det same vindauget som i figur 1.

## Slik fungerer applikasjonen

Frå start har du eit spel med enkelt utsjåande kor delar av grensesnittet manglar. Når applikasjonen er ferdig implementert (figur 2) er spelet fungerande med oppdatert grafikk og fungerande grensesnitt.



Figur 2: Skjerm bilde av den ferdige applikasjonen.

## Korleis svare på oppgåvene

Kvar oppgåve i del 3 har ein unik kode for å gjere det lettare for deg å vite kor du skal fylle inn svara dine. Koden er på formatet `<T><siffer> (TS)`, der siffera er mellom 1 og 11 ( $T1 - T11$ ). For kvar oppgåve vil ein finne to kommentarar som skal definere høvesvis begynninga og slutten av svaret ditt. Kommentrane er på formatet: `// BEGIN: TS` og `// END: TS`.

For eksempel kan ei oppgåve sjå 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 løysinga di bør du enda opp med følgjande:

```
// 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 svara dine blir ført mellom slike par av kommentarer.** Dette er for å støtte sensurmekanikken vår. Viss det allereie er skreven kode *mellom* BEGIN- og END-kommentarane til ei oppgåve i kodeskjelettet som er gitt ut, så kan du, og ofte bør du, erstatte denne koden med din eigen implementasjon. All kode som er skreven *utanfor* BEGIN- og END-kommentarane **SKAL** du la stå som den er. I oppgåve T8 er det ikkje nokon funksjonsdeklarasjon eller funksjonsdefinisjon mellom BEGIN- og END-kommentarane, så her må du skrive all kode sjølv, og det er viktig at du skriv all koden din mellom BEGIN- og END-kommentarane for å få utteljing for oppgåven.

**Merk:** Du skal **IKKJE** fjerne BEGIN- og END-kommentarane.

Dersom du synast nokon av oppgåvene er uklare kan du oppgje korleis du tolkar dei og det du antar for å løyse oppgåva som kommentarar i koden du leverer.

**Tips:** trykker ein CTRL+SHIFT+F og søker på BEGIN: får ein snarvegar til starten av alle oppgåvene lista opp i utforskervindauget så ein enkelt kan hoppe mellom oppgåvene. For å kome tilbake til det vanlege utforskervindauget kan ein trykke CTRL+SHIFT+E.

# Oppgåvene

## 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: Klassedeklarasjonar 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 frå Entity.

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

Board-klassen (Figur 3) har ein medlemsvariabel `size` som seier kor stort brettet er i én dimensjon. Medlemsfunksjonen `get_size` blir brukt i koden til å henta ut verdien til `size`, men for augeblinken returnerer `get_size` alltid 8, så brettet har alltid størrelsen  $8 \times 8$ . Endre funksjonen `get_size` så den returnerer verdien til `size`.

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

2. (10 points) **T2: Kva for spelar?**

Ein spelar blir representert av ein enum-klasse kalla `Player` (Figur 3). Vi ynskjar no ein funksjon som kan ta ein instans av `Player` som parameter og returnere ein streng som representerer namnet til spelaren. Viss spelaren er `Player::ONE` skal "ONE" bli returnert, medan viss spelaren er `Player::TWO` skal "TWO" bli returnert. Utanom dette skal funksjonen returnere "invalid".

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

- Viss spelaren er `Player::ONE`, returner "ONE".
- Viss spelaren er `Player::TWO`, returner "TWO".
- Viss spelaren verken er `Player::ONE` eller `Player::TWO`, returner "invalid".

Når oppgåve T2 er gjort skal du kunne sjå kva for spelar sin tur det er øvst til venstre på skjermen.

### 3. (15 points) T3: Innafor brettet?

For å kunne interagere med spelet må det vere mogleg å klikke på brettet. Funksjonaliteten for å klikke er for det meste allereie implementert, men éin av funksjonane må endrast. Funksjonen `inside_interaction_zone` i `BoardGame.cpp` skal returnere `true` dersom eit punkt ligg innafor sjølve brettet, men for augeblinken returnerer den alltid `true`.

Funksjonen `inside_interaction_zone` tek inn eit punkt, `point`, som parameter, og skal avgjere om `point` ligg på innsida av brettet. For å henta ut informasjon om brettet kan ein bruke funksjonen `getPosition()` som returnerer eit `TD4102::Point` som representerer posisjonen til det øvste venstre hjørnet til brettet. Funksjonane `get_width()` og `get_height()` kan brukast for å henta ut høvesvis bredden og høgda til brettet. Du kan sjå delar av definisjonen til `BoardGame` i Figur 4, og heile definisjonen finst i `BoardGame.h`.

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

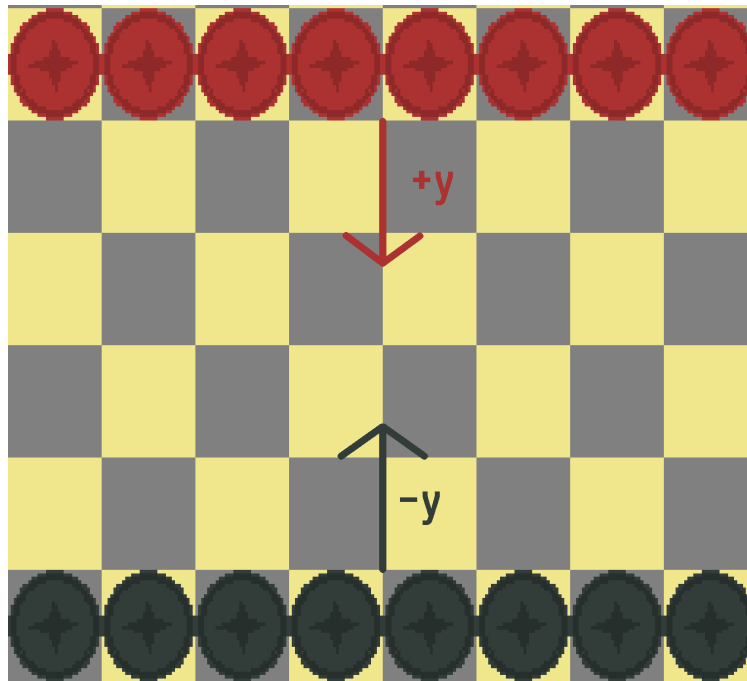
Gitt `point`, returner `true` dersom:

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

der  $b_x$  og  $b_y$  er høvesvis  $x$ - og  $y$ -koordinaten til det øvste venstre hjørnet til brettet, og  $w$  og  $h$  er høvesvis bredden og høgda til brettet.

Når oppgåve T3 er gjort vil det ikkje skje noko på skjermen dersom ein klikkar utanfor sjølve brettet.

## Visuelle detaljar og reglar (70 poeng)



**Figur 5:** Illustrasjon av bevegelsesretning for bøndene til spelar éin (øvtst) og spelar to (nedst).  $+y$  betyr ein positiv endring i  $y$ , medan  $-y$  betyr ein negativ endring i  $y$ .

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

Dette skal vere bondesjakk, men slik spelet er no kan vi flytte bøndene i kvar og ein retning, altså både opp, ned, til venstre og til høgre. Som i tradisjonell sjakk, kan bøndene kun bli flytta i éi retning, opp eller ned avhengig av spelaren. Spelaren som startar øvtst på brettet skal bevege seg nedover, medan spelaren som startar nedst på brettet skal bevege seg oppover. Riktig retning for kvar spelar er vist i Figur 5.

Vi skal no implementere funksjonen `is_valid_direction` i `Tasks.cpp`, som basert på parameterane `from`, `to` og `player` skal returnere ein boolsk verdi som seier om eit trekk frå `from` til `to` frå spelar `player` er i riktig retning. For spelar éin (`Player::ONE`) er eit trekk i riktig retning et trekk som gir positiv endring i  $y$ -koordinaten, medan det for spelar to (`Player::TWO`) er eit trekk som gir negativ endring i  $y$ -koordinaten.

Implementer funksjonen `is_valid_direction` i `Tasks.cpp`

- Returner `true` viss differansen mellom  $y$ -koordinatane til punktet `from` og `to` er positiv og `player` er `Player::ONE`.
- Returner `true` hvis differansen mellom  $y$ -koordinatane til punktet `from` og `to` er negativ og `player` er `Player::TWO`.
- Returner `false` elles.

### 5. (15 points) T5: Bilder til brikker



Brikkene blir no vist på skjermen som einsfarga raude og blå sirkclar, noko som er litt kjedelig. Vi vil derfor gjere brikkene litt finare å spela med ved å laste inn bilder for brikkene. For å hjelpe med dette finst klassen `ImageAtlas`.

Klassen `ImageAtlas` har ein medlemsfunksjon `get_image` definert i `ImageAtlas.cpp` som blir brukt til å henta ut bilder, men implementasjonen til denne funksjonen er ufullstendig. Slik implementasjonen er no returnerer funksjonen ein standard `std::shared_ptr`, men vi vil at funksjonen skal returnere ein `std::shared_ptr` til eit `TDT4102::Image`.

I `ImageAtlas::get_image` finst allereie variabelen `container`, som er eit `std::unordered_map` med nøkkeltipe `std::string` og element-type `std::shared_ptr<TDT4102::Image>`. Vi vil bruke parameteren `key` for å henta ut ein `std::shared_ptr` til eit `TDT4102::Image`. Dersom `container` ikkje inneheld eit element for nøkkelen `key`, vil vi at funksjonen skal returnere ein standard `std::shared_ptr`.

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

- Bruk parameteren `key` til å henta ut og returnere ein `std::shared_ptr` til eit `TDT4102::Image` frå `container`.
- Returner ein standard `std::shared_ptr` viss `container` **ikkje** inneheld eit element for nøkkelen `key`.

Når oppgåve T5 er gjort skal du kunne sjå at brikkene på skjermen har gått frå å sjå ut som i Figur 1 til å sjå ut som i Figur 2.

#### 6. (20 points) T6: Tell brikkene

Til venstre for brettet finst det to teljarar som skal vise kor mange brikker kvar spelar har på brettet. Desse teljarane er avhengige av funksjonen `count_chips` i `Tasks.cpp` som enda ikkje er implementert, og dei viser derfor at spelarane har 0 brikker kvar på brettet uansett korleis brettet ser ut. Funksjonen `count_chips` tek ein 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 tel antall ruter på brettet som er okkupert av spelaren `Player::ONE` og antall ruter på brettet som er okkupert av spelaren `Player::TWO`.
- Returner antall ruter på brettet som er okkupert av kvar spelar.

**Hint:** Det finst allereie eit `std::pair<int, int>` kalla `counts` i funksjonen som du kan bruke til å telja rutene som blir okkupert av kva spelar. Eit `std::pair` har to felt kalla `first` og `second` som du kan lese frå og skrive til ved å bruke notasjonen `counts.first` og `counts.second`.

Når oppgåve T6 er gjort skal du kunne sjå at antall ruter på brettet som er okkupert av spelarane blir vist i teljarane til venstre for brettet på skjermen.

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

Når ein klikkar på ei rute med ei brikke på brettet, blir ruta markert med ein annan farge enn den som var der fra før. Vi ynskjar i tillegg å vise kor brikka på den valde ruta kan bli flytta. For å oppnå dette skal vi no implementere funksjonen `BoardGame::highlight` i `BoardGame.cpp`.

For å gjere denne oppgåva må du bruke funksjonen `Rules::can_move` som er definert i `Rules.cpp`. Du treng ikkje å bry deg om implementasjonsdetaljane, men du kan sjå funksjonsdeklarasjonen i Figur 7. Denne funksjonen returnerer ein boolsk verdi basert på om eit trekk frå ruta med koordinat `from` til ruta med koordinat `to` er gyldig. Den tek inn ein referanse til brettet, koordinaten til ruta trekket startar i (`from`), koordinaten til ruta trekket sluttar i (`to`) og eit `Player`-objekt som seier kven sin tur det er (`turn`.)

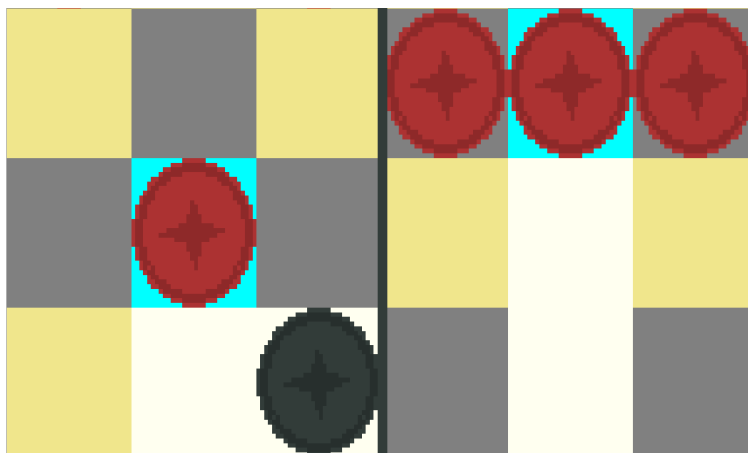
For å kunne markere alle rutene som ein spelar kan flytte ei gitt brikke til må vi sjekke alle rutene på brettet. For kvar rute på brettet skal vi bruke funksjonen `can_move` for å avgjere om eit trekk frå `selected` til ruta er gyldig og merke ruta med verdien som `can_move` returnerer.

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

**For kvar 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 ein instans av `Board`.
  - Parameteren `from` skal motta koordinaten til ruta med brikka ein ynskjar å flytte.
  - Parameteren `to` skal motta koordinaten til ruta ein sjekker om ein kan flytte til.
  - Parameteren `turn` skal motta enum-verdien for spelaren viss tur det er.

Når oppgåve T7 er gjort skal du kunne sjå gyldige trekk markert på brettet sånn som i Figur 6 når du klikker på ei rute med ei brikke.



**Figur 6:** Skjermbilde av applikasjonen etter `BoardGame::highlight` er implementert. Bildet viser to situasjonar. *Venstre:* ei brikke som kan flytte rett fram eller eliminere brikken nedanfor til høgre. *Høgre:* ei brikke som ikkje har blitt flytta før og kan flytte éi rute eller to rutar 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: Likheitsoperator for Point

Strukturen `TDT4102::Point` blir mykje brukt i denne applikasjonen. Strukturen kan du sjå under, sjølv om du kanskje allereie er godt kjent med den:

```

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

```

Vi ynskjar å kunne sjekke om to `TDT4102::Point` er like, og for å kunne gjere 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 viss punkta er like.
- `operator==` skal returnere false viss punkta er ulike.

**Merk:** Oppgåva ber deg om å skrive heile overlastinga. Hugs å skriv svaret ditt innafor `BEGIN-` og `END-`kommentarane til oppgåve T8 for å få uttelling for oppgåven.

Når oppgåve T8 er gjort vil du kunne sjå at det står "Point equality overload in use!" når du startar applikasjonen.

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

Vi ynskjer å tilby spelarane moglegheita til å tilbakestilla spelet, og kanskje til og med velje større brett å spele på. For augeblinken skjer det ingenting dersom ein trykker på `Reset`-knappen fordi funksjonen som som blir kalla når `Reset`-knappen blir trykt på, `reset.button_callback`, ikkje er implementert. Dette skal vi fikse no.

Det er nokon detaljar som er viktige å ha i bakhovudet når du jobbar med denne oppgåva. Det finst to globale unike peikarar å forhalde seg til. `list` er ein `std::unique_ptr` til eit `TDT4102::DropDownList`-objekt, og representerer nedtrekksmenyen til venstre for `Reset`-knappen. `game_ptr` er ein `std::unique_ptr` til eit `BoardGame`-objekt, og representerer brettet ein ser på skjermen. Begge definisjonane til dei global unike peikarane finst i `main.cpp`.

Når ein trykker på Reset-knappen skal objektet som `game_ptr` peikar til bli skifta ut med ein ny instans av `BoardGame`. Vi kan derimot ikkje berre lage ein ny instans av `BoardGame`, sidan posisjonen, bredden og høgda da ikkje nødvendigvis blir dei same som før. Det første vi skal gjere er derfor å henta ut og lagre den noverande posisjonen, bredden og høgda så vi kan oppdatere verdiane til den nye `BoardGame`-instansen med dei same verdiane som den gamle `BoardGame`-instansen hadde. Storleiken til den nye `BoardGame`-instansen skal vere gitt av verdien i nedtrekksmenyen til venstre for Reset-knappen.

Implementer `reset_button_callback` i `Tasks.cpp`.

**Funksjonen skal:**

- Lagre posisjonen, bredden og høgda til `BoardGame`-instansen den globale variabelen `game_ptr` peikar til.
- Henta ut verdien til `TDT4102::DropDownList`-instansen som den globale variabelen `list` peikar til.
- Konvertere strengen henta ut i det førre steget til eit heiltal.
- Overskrive `game_ptr` sin `BoardGame`-instans med ein ny `BoardGame`-instans som har storleik lik verdien som blei henta ut og konvertert i dei to førre stega.
- Setje bredden, høgda og posisjonen til den nye `BoardGame`-instansen lik verdiane som blei henta ut i første steg.

**Hint 1 :** Du kan bruke konstruktøren til `BoardGame` med ein heiltalsparameter for å lage eit heilt nytt `BoardGame`-objekt med alle brikkene satt på plass.

**Hint 2:** Medlemsfunksjonane til `BoardGame` er arva frå `Entity` (Figur 4).

Når oppgåve T9 er gjort skal ein kunne bruke Reset-knappen til å tilbakestille spelet. Det nye brettet skal ha same posisjon, bredde og høgde som det førre brettet, og storleik gitt av nedtrekksmenyen til venstre for Reset-knappen.

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

No skal vi implementere ein kopikonstruktør for klassen `BoardGame` som tek utgangspunkt i eit `Board`-objekt. Målet med oppgåva er å sikre at kopikonstruktøren initialiserer et nytt `BoardGame`-objekt som tek i bruk brettet (`Board`) den mottok.

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

**Kopikonstruktøren skal sørge for at:**

- Det blir spelar éin (`Player::ONE`) sin tur (sett verdien til `turn`).
- Vinnaren blir tilbakestilt til `Player::NONE` (sett verdien til `winner`).
- Spelet blir sett i gang ved å setje `enabled` til `true`.
- `highlighted` vektoren blir tømt.
- `highlighted` vektoren sin storleik blir tilpassa storleiken til `Board`-objektet den mottok.

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

Vi ynskjer no å gjere det mogleg å ta opp igjen eit uferdig spel. For å gjere det må vi både kunne lagre tilstanden til eit spel i ei fil, og laste inn eit spel frå ei fil. Funksjonen for å lagre tilstanden til eit spel er ikkje implementert, men det finst eit "uferdig" spel lagra som tekst i fila `state`.

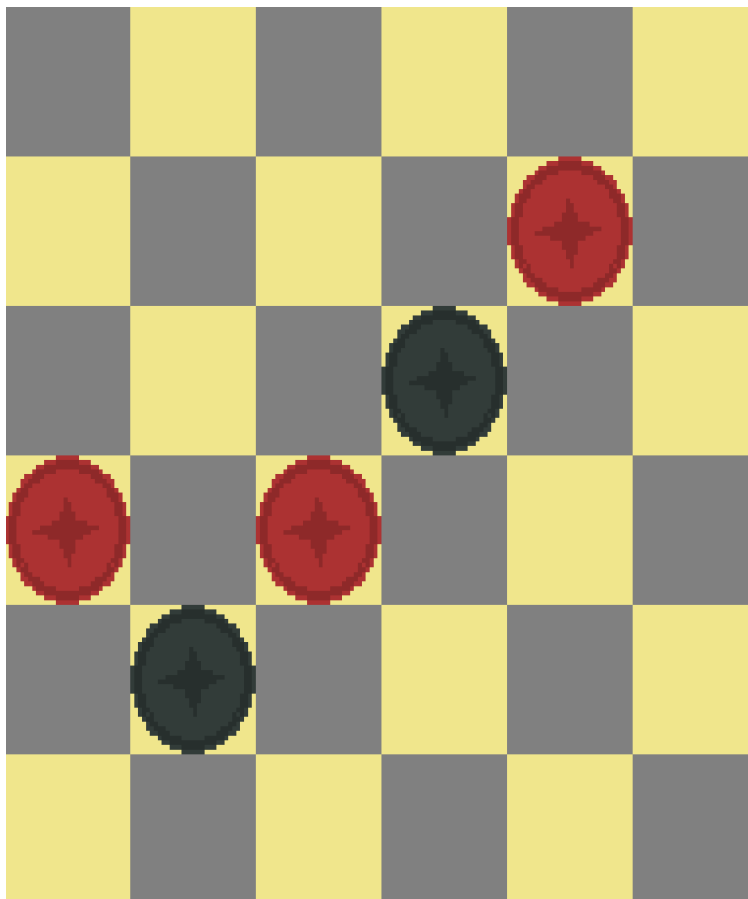
Du skal implementere funksjonen `load_board` i `Board.cpp` som les inn ein speltilstand frå ei fil og opprettar eit `Board`-objekt som representerer tilstanden beskrive i fila. Fila inneheld informasjon om storleiken til brettet og tilstanden til kvar rute. Første linje i fila er eit heiltal  $N$  som representerer storleiken til brettet i kvar dimensjon ( $N \times N$ ). Dei neste  $N$  linjene er  $N$  heiltal separert av mellomrom som representerer tilstanden til kvar rute på brettet. Ei rute kan vere tom (0), innehalde ei brikke frå spelar éin (1), eller innehalde ei brikke frå spelar to (2). Eit eksempel på korleis dette ser ut kan sest i fila `state` eller i Figur 9.

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

- Opne fila gitt av parameteren `path`.
- Kontroller at fila ble opna og kast ein `std::runtime_error` med ei passande feilmelding viss den ikkje blei det.
- Les den første linja i fila for å henta ut storleiken til brettet og initialiser eit `Board`-objekt med denne storleiken.
- For kvar av dei neste  $N$  linjene:
  - Les linja.
  - Konverter kvart heiltal 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 fila når all data er lest og prosessert.
- Returner det nye `Board`-objektet.

**Hint:** Bruk `cell.at` funksjonen i `Board`-klassen for å gjere det lettere å finne riktig rute. `cell.at` tek inn to heiltal  $x$  og  $y$ , og returnerer ein referanse til ruta (`Tile &`) med koordinaten  $(x,y)$ .

Når oppgåve T11 er gjort skal du sjå det same brettet som i Figur 8 på skjermen.



**Figur 8:** Korrekt innlasta fil for oppgåve 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 eit brett med storleik  $6 \times 6$ .

**i Oppgave/utdelt kode del 3**

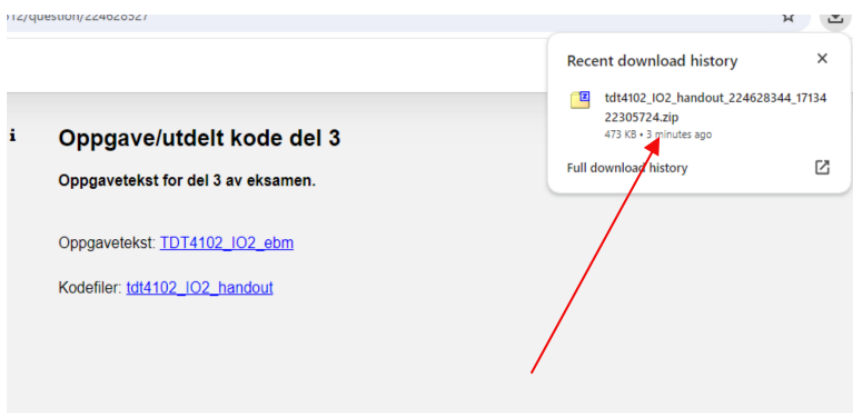
Oppgavetekst for del 3 av eksamen.

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

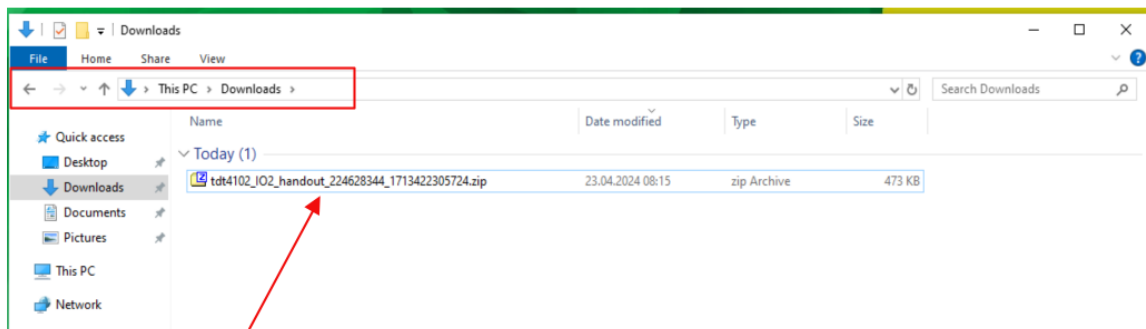
 tdt4102\_IO2\_handout\_224628344\_17134\_22305724.zip

473 KB • 3 minutes ago

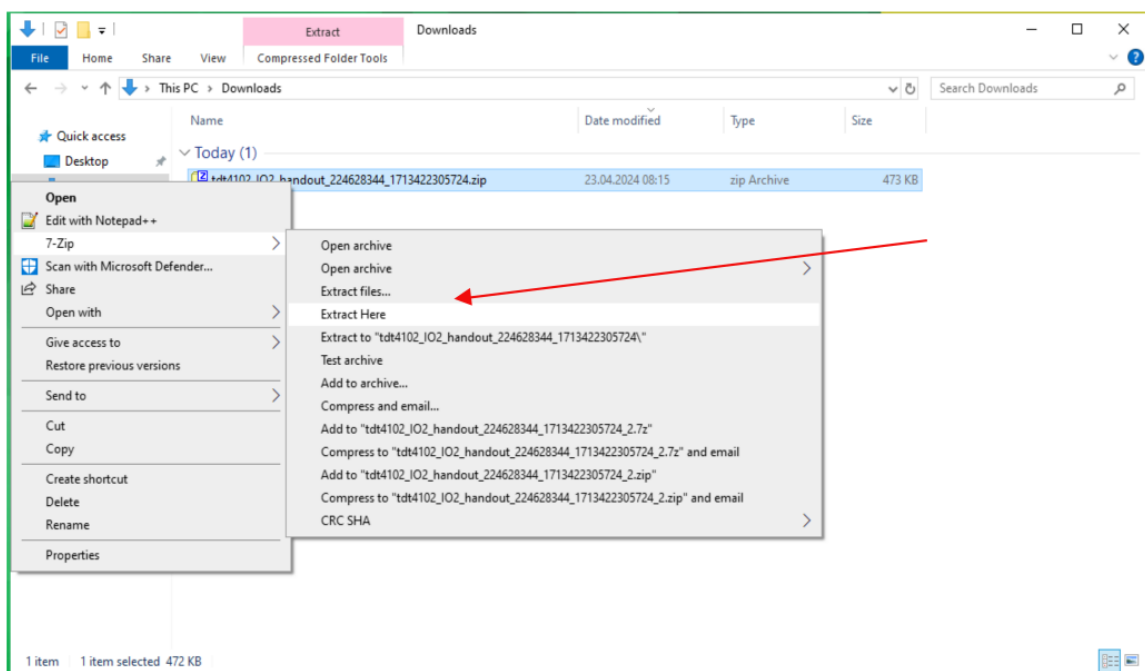
Full download history

Side 15 av 20

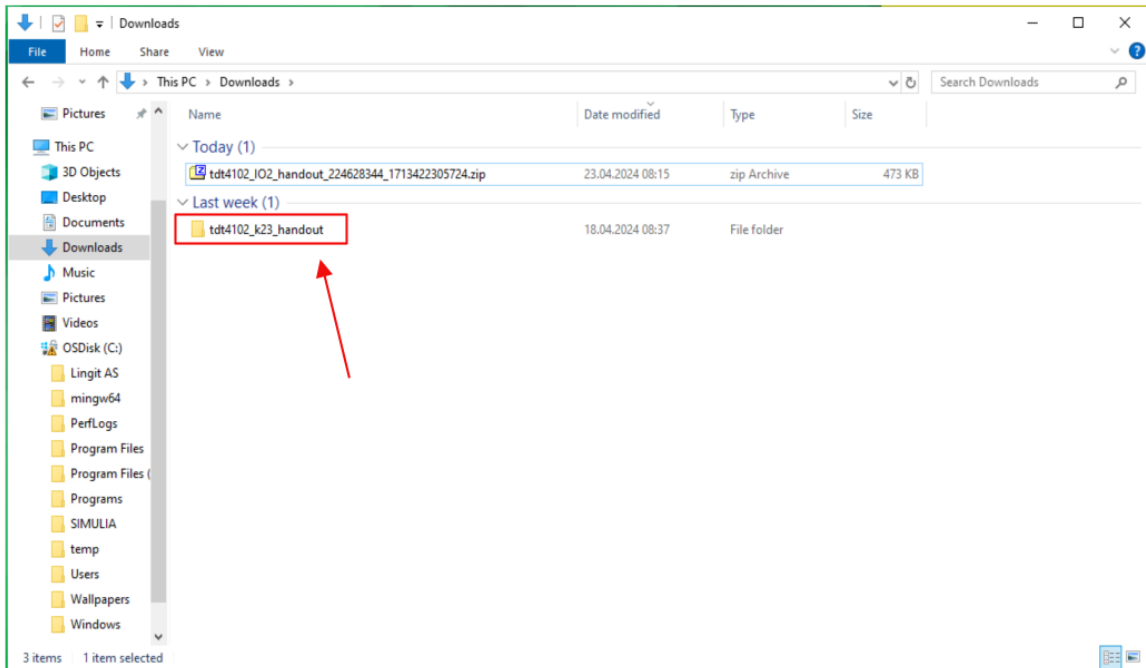
(b) Steg 1



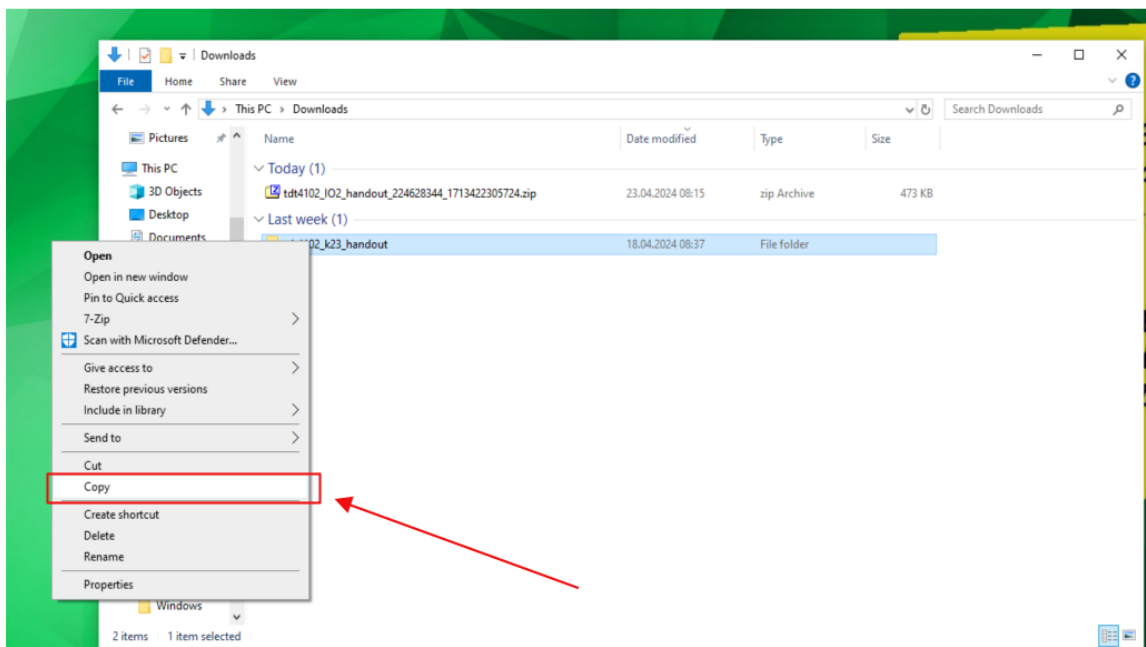
(c) Steg 2



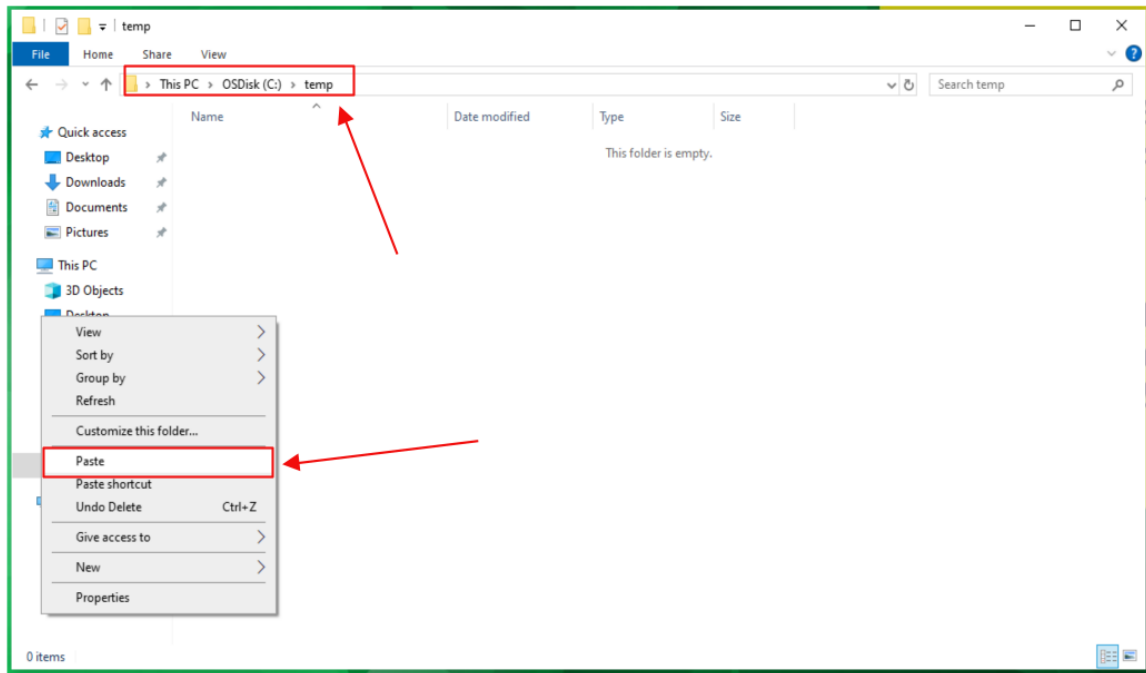




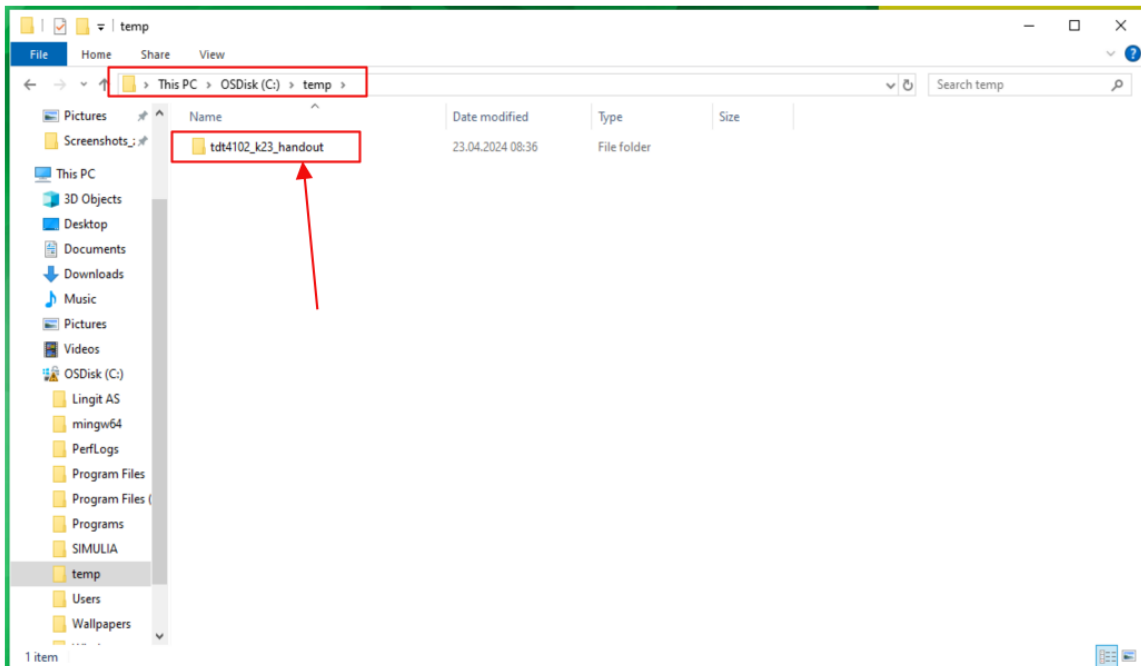
(e) Steg 3

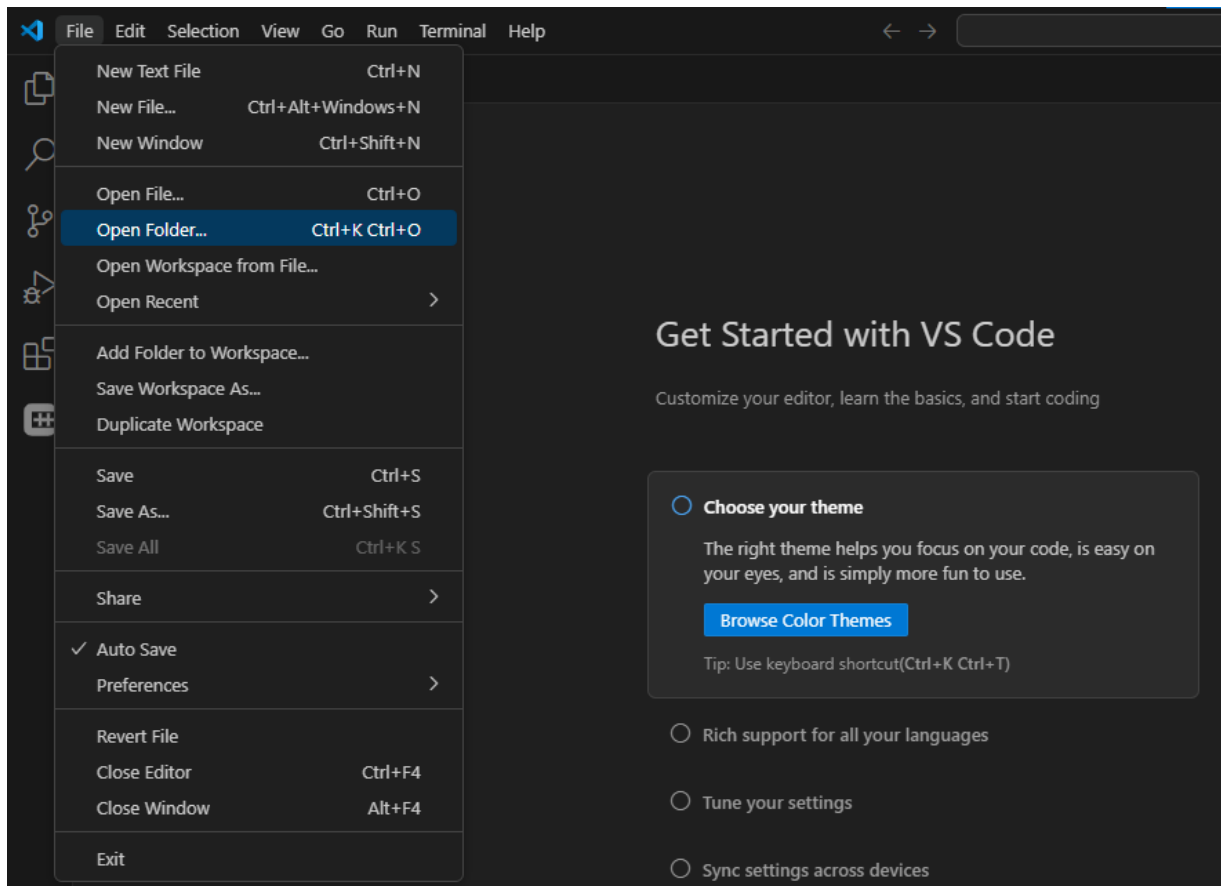


(f) Steg 4

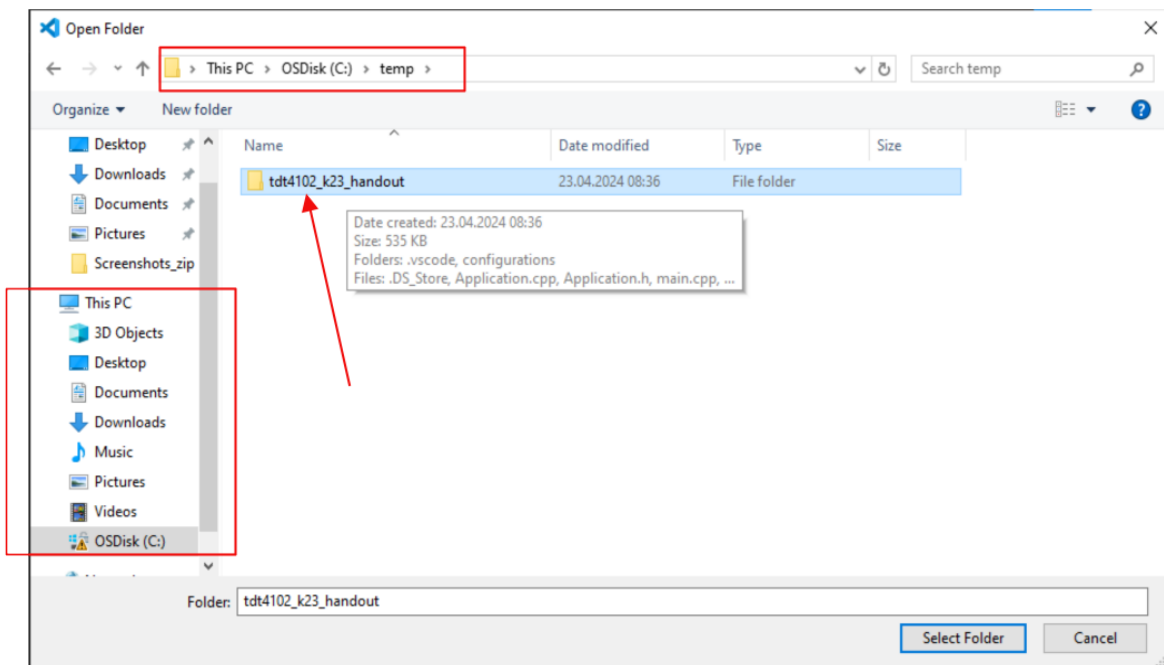


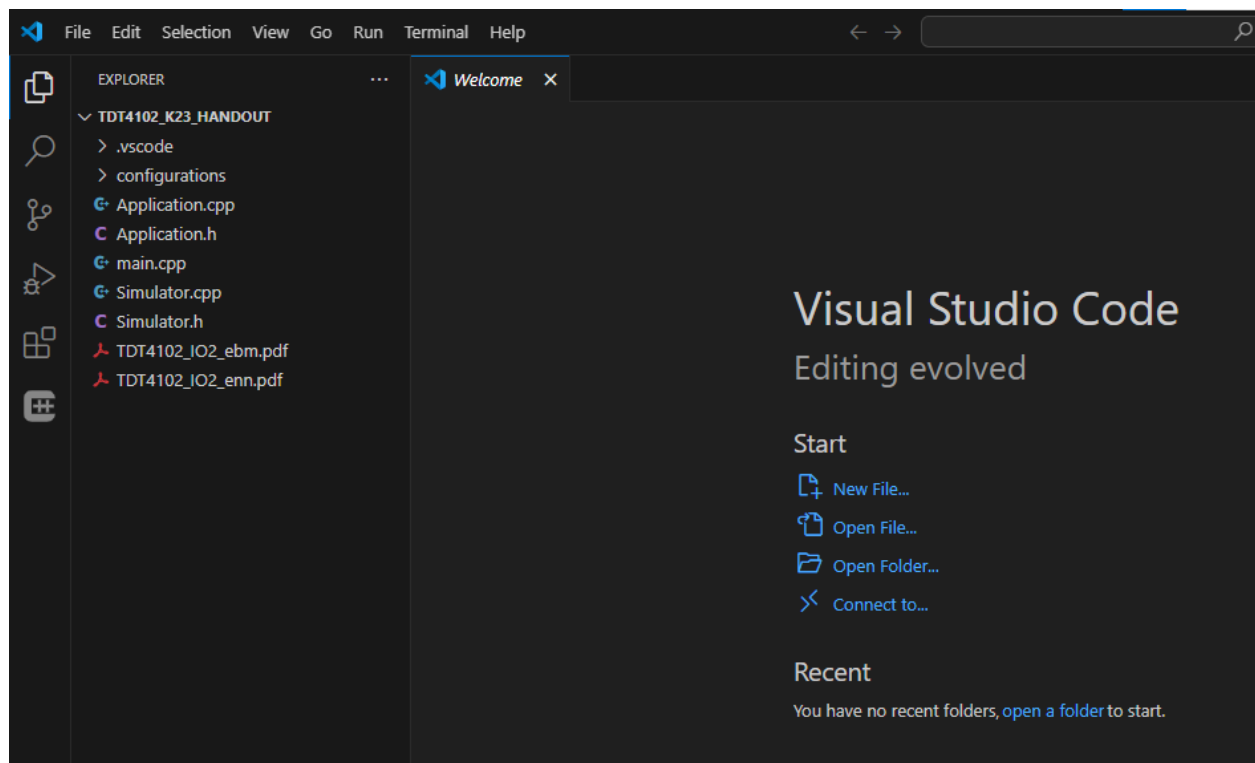
(g) Steg 4





(i) Steg 5





(k) Steg 5

**Figur 10:** Nedlasting og oppsett av den utdelte koden.