Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering
Vår 2021

Øving 12

Frist: 2021-04-22

Aktuelle temaer for denne øvingen:

- Klasser, arv og GUI

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Teorioppgaver besvares med kommentarer i kildekoden slik at læringsassistenten enkelt finner svaret ved godkjenning.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Det anbefales å benytte en programmeringsomgivelse(IDE) slik som Visual Studio Code.

Anbefalt lesestoff:

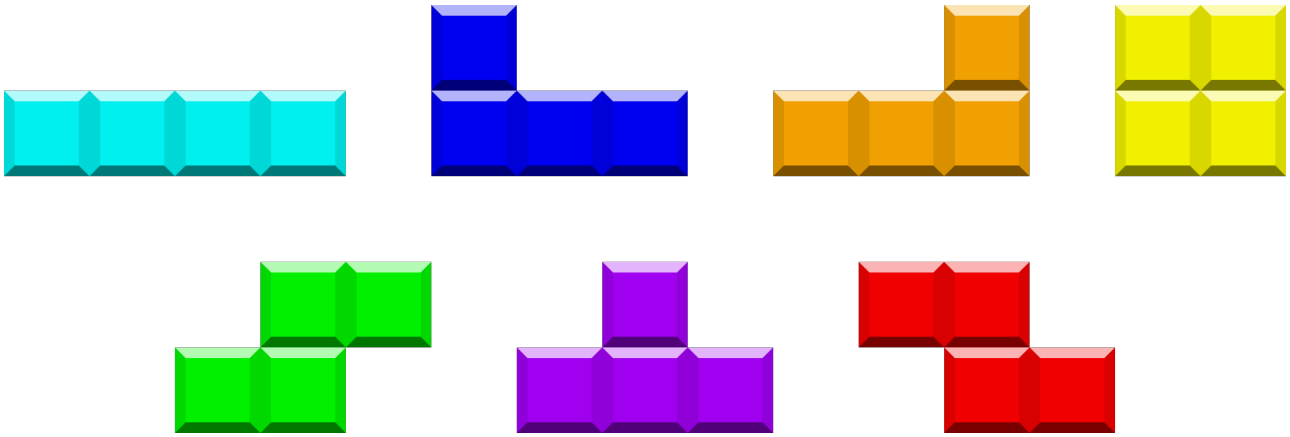
- Tetris

Bakgrunn for oppgaven

I denne øvingen skal vi implementere Tetris. Dersom du kjenner til reglene i Tetris fra før kan du hoppe over denne seksjonen.

Tetris ble oppfunnet så tidlig som i 1984, og er et av de mest populære spillene noensinne. Tetris består av et rutenett med blokkformer. Når spillet starter genereres det en Tetrisform, heretter kalt tetromino, som sakte men sikkert går mot bunnen av Tetris-vinduet. Tetrominoen fortsetter helt til den enten treffer bunnen av Tetris-vinduet, eller en eksisterende Tetris-blokk. Da blir Tetrominoen en del av blokkene i spillvinduet, og det genereres en ny, styrbar tetromino, som faller sakte mot bakken.

Alle tetrominoer har fire blokker som henger sammen. Det finnes syv ulike former, i tillegg til rotasjonene av disse formene. Formene kan du se på bildet under.



De syv tetrominoene i Tetris. Tetrominoene navngis ofte ut ifra hvilken bokstav de ligner på. Konvensjonen er, fra øverst venstre til nederst høyre: I, J, L, O, S, T, Z. Noen av blokkene må roteres for å ligne.

For å få blokkene til å passe sammen kan man både rotere tetrominoene med og mot klokken, samt flytte de til begge sider eller nedover. Man kan ikke flytte en Tetromino oppover vinduet. Spillet fortsetter helt til spilleren ikke klarer å plassere en tetromino slik at det kan genereres en ny en uten at den krasjer med en eksisterende blokk. Hvis en rad blir helt full av blokker, forsvinner alle blokkene på raden, og blokkene over faller ned en rad i vinduet. Slik kan man sørge for at haugen med Tetrisblokker aldri blir så høy at man taper. Jo lenger man holder på, jo flere poeng får man.

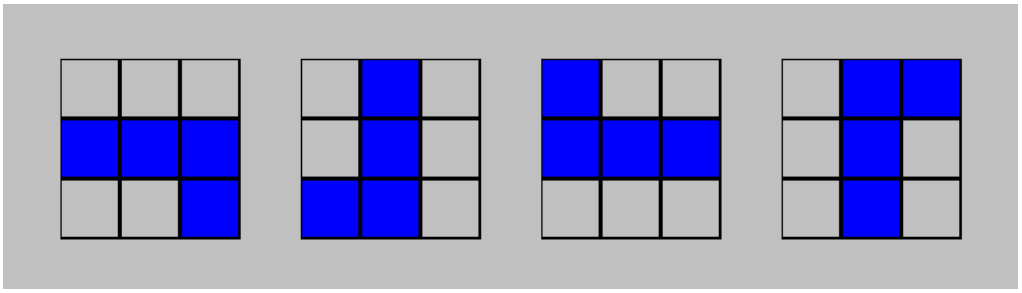
Det finnes ulike regler for hvordan spillet fungerer, blant annet hvordan rotasjon er implementert. Det finnes også ofte ekstra funksjonalitet, som at man kan se de tre neste tetrominoene for å kunne planlegge. I denne øvingen kommer vi til å implementere en veldig enkel versjon, men du står fritt til å prøve på mer komplisert logikk eller flere funksjoner dersom du ønsker dette.

Hvis spillereglene ikke kom tydelig fram av denne teksten, kan man lese mer fra lenken som er lagt inn under anbefalt lesestoff eller prøve spillet selv her: tetris.com.

1 Klassen Tetromino (25%)

I den utdelte koden er det definert en klasse `Tetromino`. En tetromino består av fire blokker, der posisjonen til de ulike blokkene kan representeres på et rutenett i matriseform. For å tegne blokkene bruker vi `Tetromino`-objekter. Matrisen skal implementeres med en dobbeltvektor, `vector<vector<Tetrominotype>>`.

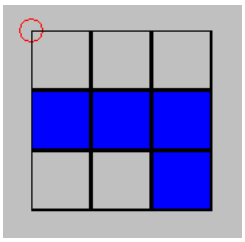
Matrisestørrelsen defineres ut ifra hva slags tetromino som lages. En J-tetromino kan for eksempel representeres ved en 3x3-matrise, mens en O-tetromino bare trenger en 2x2-matrise. Vi definerer derfor en medlemsvariabel, `matrixSize` som holder denne verdien. Bildet av alle orienteringene til en J-tetromino kan sees under.



J-tetrominoen i alle mulige rotasjonsposisjoner. Vi kan representere alle posisjonene i et 3x3 rutenett. Tetrominoen kalles J-tetromino fordi den ligner på en J i den andre orienteringen fra venstre.

Fra figuren ser vi at det er ni ruter i matrisen, men bare fire blokker. Det er derfor ikke alle posisjoner i matrisen som er tilknyttet en blokk. Vi kan representere en tom rute ved å bruke verdien `TetrominoType::NONE`.

I tillegg trenger vi å vite hvor på `AnimationWindow` vi skal tegne matrisen vår. Vi har derfor definert `Point`-variabelen `topLeftCorner`. Denne holder posisjonen til hjørnet øverst til venstre i matrisen. Dette hjørnet er valgt ettersom `Tetromino`-objekter tegnes med det øverste venstre hjørnet som utgangspunkt.



Den røde sirkelen markerer `topLeftCorner` i matrisen. Matrisen tegnes med dette punktet som utgangspunkt.

Medlemsvariabelen `blockSize` er størrelsen på tetrominoblokkene. Denne er laget i en passe størrelse, men kan endres på hvis ønskelig.

a) Fullfør initialiseringen av medlemsvariablen `matrixSize` i konstruktøren `Tetromino::Tetromino(Point startingPoint, TetrominoType tetType)`.

Tips: Du må bruke map-et `initialMatrixMap` som er definert i den utdelte koden. Per nå inneholder map-et bare matrisen til én tetromino, J. Vi skal senere legge til resten. `static_cast` kan være nyttig. J-tetrominoet skal ha en matrixSize på 3.

b) Initialiser tetromino-matrisen `blockMatrix` i konstruktøren.

Matrisen skal holde `TetrominoType`-objektene som er blokkene i tetrominoet. Bruk map-et `initialMatrixMap` for hvilke elementer som skal være en blokk, og hvilke som skal være tomme, altså `TetrominoType::NONE`. Tallet 1 i `initialMatrixMap` symboliserer en blokk.

Legg merke til at matrisene vi bruker er på formen row × column, der rows er vannrette rader, og columns er loddrette kolonner. Dette betyr at row hører til y-aksen i tegningen, og column hører til x-aksen. På denne måten vil en J-tetromino representeres slik:

NONE	NONE	NONE
J	J	J
NONE	NONE	J

Husk å initialisere elementene som ikke er en blokk til `TetrominoType::NONE`, slik at du får en matrixSize × matrixSize stor matrise. Hvis ikke vil du oppleve "out of range"-feil senere i øvingen.

c) Lag defaultkonstruktøren `Tetromino::Tetromino()`.

Initialiser `topLeftCorner` til {0,0}, og `matrixSize` til 0. Defaultkonstruktøren skal ikke lage noen `Tetromino`-objekter.

2 Bevegelige Tetrominoer (20%)

a) I den utdelte koden finnes det to medlemsfunksjoner til `Tetromino`, `Tetromino::rotateClockwise()`, og `Tetromino::rotateCounterClockwise()`. Disse roterer matrisen `blockMatrix` 90 grader hendholdsvis med og mot klokken. Funksjonene er ikke komplette, ettersom de ikke oppdaterer rotasjonen i grafikken. Vi skal derfor lage en medlemsfunksjon som oppdaterer posisjonen til blokkene i grafikken.

Lag medlemsfunksjonene `Tetrimino::moveDown()`, `Tetrimino::moveLeft()` og `Tetrimino::moveRight()`.

Funksjonene skal flytte hele tetrominoet en blokk i den respektive retningen. Test funksjonene i `main()`.

Hint: Ettersom orienteringen ikke endrer seg når man flytter hele objektet, trenger man ikke å endre matrisen `blockMatrix` i disse funksjonene. Bruk eksisterende medlemsfunksjoner og variabler. Husk særlig at du har tilgang på det øverste hjørnet til venstre i matrisen gjennom medlemsvariabelen `topLeftCorner`. Disse funksjonene kan lages svært korte.

b) **Lag medlemsfunskjonen `Tetromino::blockExist(int row, int column)`.**

Funksjonen skal ta inn rad og kolonne i matrisen, og returnere en boolsk verdi på om det finnes en blokk på denne plassen eller ikke. Husk å sjekke om indeksen som sendes inn er utenfor matrisen.»

c) **Lag en medlemsfunskjon `Tetromino::getBlock(int row, int column)`.**

Funksjonen skal ta inn rad og kolonne i matrisen, og returnere TetrominoType-en til elementet.

d) **Lag en medlemsfunskjon `Tetromino::getMatrixSize()`.**

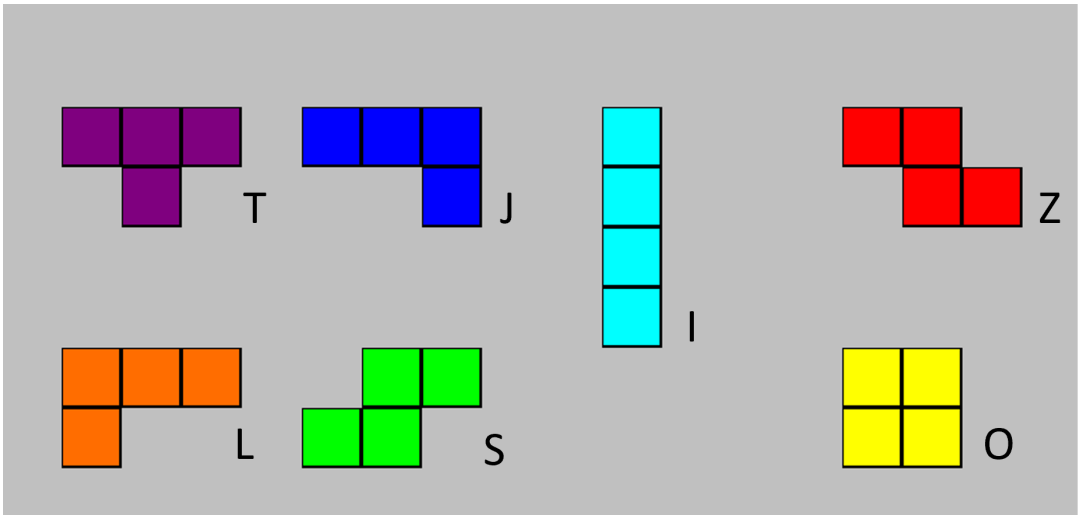
Funksjonen skal returnere matrisestørrelsen til tetrominoen.

e) **Lag en medlemsfunskjon `Tetromino::getPosition()`.**

Funksjonen skal returnere det øverste venstre hjørnet.

f) **Lag de seks andre tetrominoene, ved å oppdatere det utdelte map-et `initialMatrixMap`.**

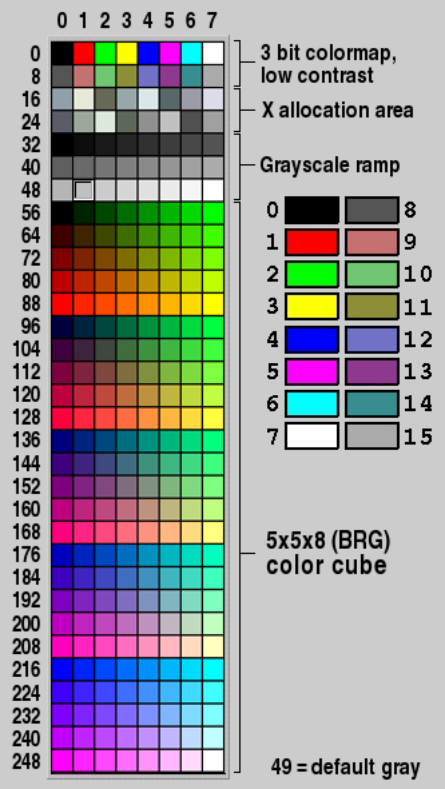
Husk at matrisene kan ha forskjellige størrelser. Gi tetrominoene ulike farger. For å enkelt gi tetrominoene farger kan du lage et map på samme måte som `initialMatrixMap`.



Inspirasjon til de syv tetrominoene.

Nyttig å vite: Farger i FLTK

Vist i bildet under er fargekartet til FLTK. Funskjonen `draw_Rectangle` kan ta inn en int-verdi som siste argument, og figuren får da fargen fra fargekartet. For eksempel vil funksjonskallet `draw_rectangle(Point, width, height, 103)` gi rektanglet en lysegrønn farge.



Fargene i FLTK.

3 Klassen TetrisWindow (25%)

I den utdelte koden er det et skjelett til `TetrisWindow`-klassen. Den utdelte koden vil bli forklart etter hvert som den blir relevant. En del av den utdelte funksjonaliteten er ikke direkte pensum, men er laget for å få spillet til å fungere med bruker-input.

a) Fullfør den utdelte koden, ved å definere medlemsvariabler og konstruktør.
Definer konstruktøren til `TetrisWindow`.
Klassen skal ha følgende medlemsvariabler:

- `vector<vector<TetrominoType>> gridMatrix`
  - Medlemsvariabelen skal representere spillvinduet som en matrise med `TetrominoType`-objekter på samme måte som i `Tetromino`-klassen.
- `Tetromino currentTetromino`
  - Medlemsvariabelen skal holde det nåværende `Tetromino`-objektet som styres

Du bør også definere konstanter for hvor langt og stort vinduet skal være i antall blokker, størrelsen på hver blokk, og et startpunkt i vinduet for de genererte tetrominoene. Startpunktet skal tilsvare vindusposisjonen til tetrominoen. Merk at klassen arver fra `AnimationWindow`.

b) Lag medlemsfunksjonen `TetrisWindow::generateRandomTetromino()`.
Funksjonen skal oppdatere medlemsvariablen `currentTetromino` med et nytt, tilfeldig `Tetromino`-objekt. Kall funksjonen fra konstruktøren for å generere et tilfeldig `Tetromino`-objekt når vinduet lages.

c) Lag medlemsfunksjonen `TetrisWindow::drawCurrentTetromino()`. Funksjonen skal tegne blokkene til `currentTetromino` ved hjelp av `draw_rectangle()`-funksjonen. Kall funksjonen på kommentert plass i den allerede definerte `run()`-funksjonen. Test funksjonen ved å opprette et `TetrisWindow` i main og kalle `run()` på det.

d) I den utdelte koden finnes det en funksjon som heter `handleInput()`. Denne skal sjekke om en gitt tast er trykket inn, og skrive til terminalen basert på hvilken tast det er snakk om. Kjør programmet og sjekk at det skrives til terminal når du trykker på de aktuelle tastene: bokstaven "z" og "pil opp".

e) Endre på `handleInput()`, for å styre `currentTetromino` med piltastene.
Man skal kunne rotere tetrominoen, og flytte den til høyre, venste eller nedover. Du kan fjerne koden som skriver til terminalen, da denne bare er et eksempel. Du står fritt til å velge hvilke taster du kobler til hvilke handlinger. I noen tetrisversjoner bruker man z-tasten for å rotere mot klokken og opp-tasten for å rotere med klokken.

Tips: Det kan være lurt å begrense hvor langt til høyre, venstre og ned tetrominoen kan gå. Hvis ikke riskerer du å aksessere utenfor vektorene i matrisen, og få kjøretidsfeil. For å gjøre det enkelt kan du begrense hvor langt selve matrisen kan gå i de ulike retningene. Dette vil ikke fungere perfekt, men vi skal lage en bedre løsning senere i øvingen.
Husk at matrisen begynner i den øverste venstre hjørnet.

f) Lag medlemsfunksjonen `TetrisWindow::moveTetrominoDown()`.
Funksjonen skal flytte det styrbare `Tetromino`-objektet ned ett hakk for hvert funksjonskall.
Tips: Ikke la tetrominoen kunne gå nedenfor vinduet, av samme grunn som den forrige deloppgaven.

g) Vi har lyst til at `TetrisWindow::moveTetrominoDown()` skal kalles i jevne intervaller, slik at tetrominoen faller sakte mot bunnen av vinduet. Vi må derfor kalle funksjonen i hovedløkken til programmet, den allerede definerte medlemsfunksjonen
`TetrisWindow::run()`.
Kall `moveTetrominoDown()` på den kommenterte plassen i `run()`.
Sjekk at tetrominoen faller mot bunnen av spillvinduet.

4 Spill-logikk (30%)

Nå som vi kan styre en tetromino gjenstår det bare å lage spillogikken. Denne oppgaven er et forslag på hvordan man kan gå frem, men hvis du ønsker kan du prøve helt eller delvis selv.

a) Når man treffer bunnen eller en annen blokk under i Tetris, skal tetrominoen stoppe opp, og det skal genereres en ny tetromino. Man må da sørge for at blokkene i den gamle tetrominoen blir en del av rutenettet til Tetris-vinduet.

Lag medlemsfunksjonen `TetrisWindow::fastenTetromino()`.

Funksjonen skal overføre blokkene fra den styrbare tetrominoen til Tetris-rutenettet, altså i `Tetris::gridMatrix`. Du kan for eksempel teste funksjonen ved å koble den til en tast i `handleInput()`.

b) **Lag medlemsfunksjonen `TetrisWindow::drawGridMatrix()`.**

Funksjonen skal tegne de festede tetrominoene fra `gridMatrix` til vinduet. Denne skal kalles for hver iterasjon av `while`-loopen i `run()`.

c) Når en tetromino treffer en hindring skal den stoppe opp. Hindringer kan både være andre blokker eller veggene i vinduet. For å gjøre det så enkelt som mulig skal vi la tetrominoen krasje og handle ut ifra det. Vi trenger derfor funksjonalitet for å oppdage når den styrbare tetrominoen krasjer.

Lag medlemsfunksjonen `TetrisWindow::hasCrashed()`.

Funksjonen skal returnere `true` dersom den styrbare tetrominoen har krasjet, og `false` hvis ikke. Funksjonen skal både teste om tetrominoen har krasjet i andre blokker, og veggene/bunnen i vinduet. Du kan lage hjelpefunksjoner for de enkelte testene hvis du ønsker. Test denne funksjonen skikkelig for å ikke få obskure feil i senere oppgaver.

Du kan fjerne andre tester for vegger og bunn du har laget tidligere.

d) **Endre funksjonaliteten i `TetrisWindow::handleInput()`, for å hindre ulovlige flytt.**

Dette betyr at tetrominoen skal stoppe når den møter en hindring. Den enkleste måten å gjøre dette på er å utføre den motsatte handlingen når man detekterer et krasj. For eksempel vil et flytt til venstre oppheve et ulovlig flytt til høyre.

Det er ulike måter å håndtere en ulovlig rotasjon på. Det enkleste er å hindre rotasjonen. I mange implementasjoner av Tetris har man imidlertid det som kalles "Wall kick". Her "hopper" tetrominoen vekk fra veggen, som gjør at den kan roteres. Du kan lese mer her. Du kan selv velge hvilken rotasjonsfunksjonalitet du implementerer.

e) **Endre funksjonaliteten i `TetrisWindow::moveTetrominoDown()`.**

Her må du lage logikk for hva som skjer når tetrominoen treffer bunnen eller en annen blokk. Her skal den gamle gamle tetrominoen overføres til Tetris-vinduet, og det skal genereres en ny, tilfeldig tetromino.

f) For at tårnet med Tetris-blokker ikke skal vokse evig, må det være mulig for blokker å forsvinne. Alle blokkene i en rad skal forsvinne hvis raden er helt full, og alle blokkene over skal da falle ned ett hakk.

Lag medlemsfunksjonen `TetrisWindow::removeFullRows()`.

Funksjonen skal detektere om det er fulle rader i Tetris-matrisen, `TetrisWindow::GridMatrix`, og slette eventuelle fulle rader. Etterpå skal alle blokkene over slettede rader i matrisen falle ned ett hakk for hver slettede rad. Kall funksjonen et passende sted i koden.

g) **Lag funksjonalitet for å sjekke om du har tapt.**

I Tetris taper man hvis nye tetrominoer som genereres automatisk krasjer med eksisterende Tetris-blokker i vinduet. Her kan du gjenbruke eksisterende funksjonalitet. Hvis man taper må man informere spilleren om at man har tapt. Gjør dette med et pop-up vindu på skjermen. Det kan også være lurt å stoppe genereringen av nye tetrominoer hvis man allerede har tapt.

5 Mer funksjonalitet (Frivillig)

a) I Tetris pleier det å være noe som heter "Hard Drop". Dette betyr at man kan slippe en blokk helt ned til den treffer bunnen eller en annen blokk, bare med ett tastetrykk. Du kan for eksempel bruke bokstaven X for å gjøre dette.

Oppdater funksjonaliteten i `Tetris::handleInput()` med "Hard Drop".

b) Det er laget mange forskjellige poengsystem i Tetris gjennom tidene. Felles for de fleste er at man får poeng for å fjerne fulle linjer med blokker, og poeng etter hvor mange linjer nedover man "Hard Drop"-er en tetromino. Ofte får man mer poeng jo flere linjer man fjerner av gangen. Et forslag på poeng er 40 for én linje, 100 for to, 300 for tre og 1200 for fire linjer, i tillegg til ett poeng for hver linje man "Hard Drop"-er.

Lag funksjonalitet for telle poeng, og vise poengene i vinduet.

c) I enspiller-Tetris spiller man som regel mot sin egen High Score.

Lag funksjonalitet for å lagre High Score, og vise den i vinduet.

Her må du lage lage funksjonalitet for å skrive High Score-verdien til og fra fil.

d) I Tetris er det en fordel å slippe å måtte restarte programmet hvis man taper.

Lag funksjonalitet for å kunne spille på nytt dersom du taper.

Koble funksjonaliteten til en knapp eller til tastaturet. Hvis du bruker knapper kan du få problemer med at du mister kontrollen(fokus) over tastene etter du har trykket på knappene. Du kan få tilbake tastaturfokus ved å kalle `Fl::focus(this)` i medlemsfunksjonene knappene er linket til. Du kan for eksempel også lage en pauseknapp, her er det bare fantasien som setter grenser!