Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2022

Øving Inspera

Praktisk:

Det er veldig viktig at du leser det praktiske før du begynner på oppgavene.
Alle oppgavene i del 3 er satt opp slik at de skal besvares i eksisterende funksjoner i .cpp-filene i den utdelte koden. I filene du skal skrive i vil du finne to kommentarer for hver oppgave, som definerer begynnelsen og slutten på koden du skal føre inn. Kommentarene er på formatet

`// BEGIN: 1a` og `// END: 1a`

Det er veldig viktig at alle svarene dine er skrevet mellom slike kommentar-par. Kode utenfor et slikt par blir ikke vurdert. BEGIN- og END-kommentarene skal **IKKE** fjernes!

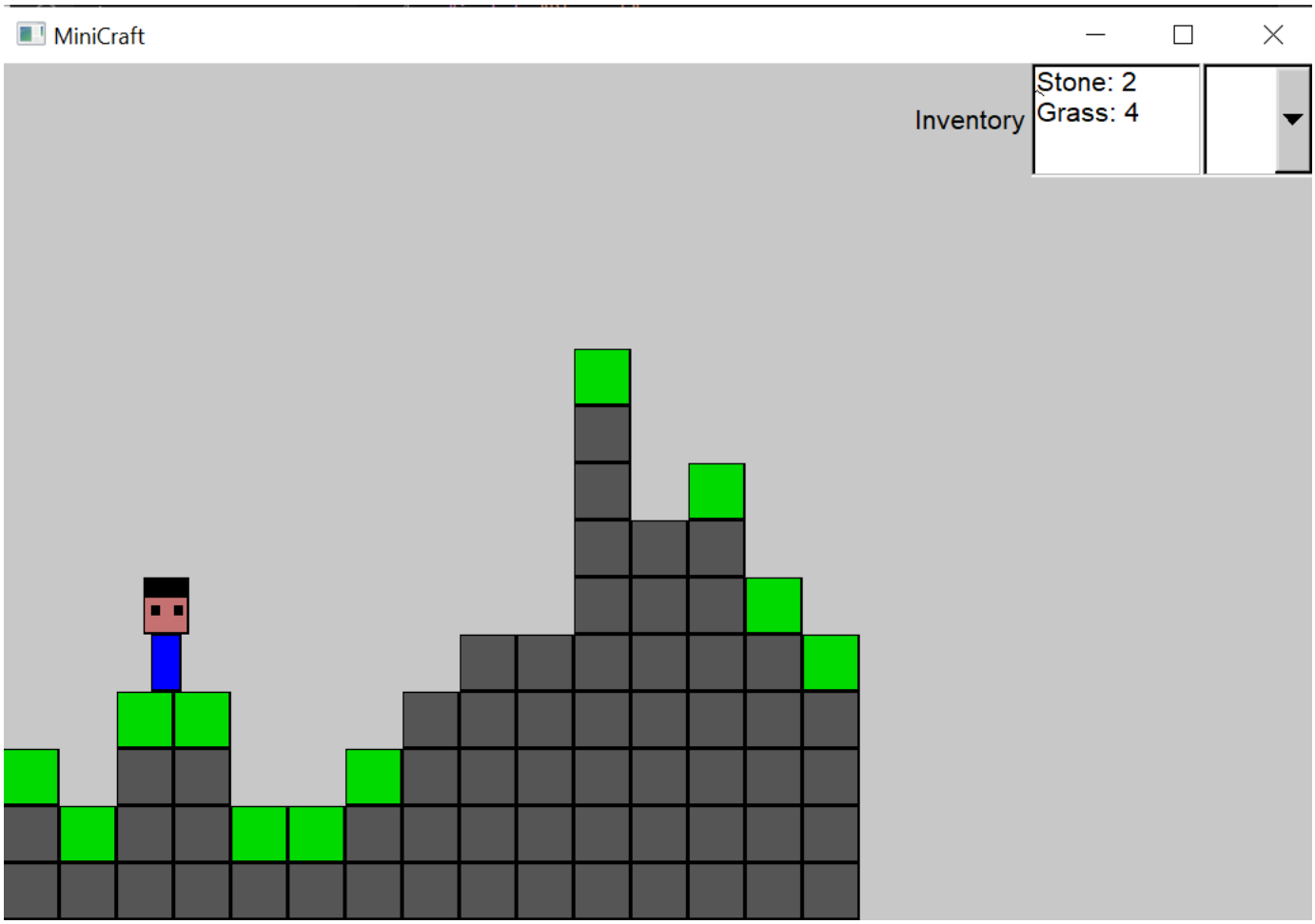
Den utdelte koden kan hentes på en av to måter:

- *Alternativ 1*: Last ned .zip-filen med utdelt kode som ligger på Inspera. Pakk denne ut på ønsket sted på PC-en din, og åpne deretter den utpakkede mappen i VSCode (slik du er vant til å åpne prosjektmapper fra øvingsopplegget). Pass på at du kun har ett nivå i mappen din i VS Code (det kommer til å bli krøll hvis kodefilene ikke ligger i den ytterste mappen).
- *Alternativ 2*: Last inn den utdelte koden via fagets VS Code extension (slik du er vant med fra øvingsopplegget). Da må du først lage en tom mappe på ønsket sted på PC-en din, og deretter åpne mappen i VS Code. Du henter den utdelte koden ved å ta:
  cmd/ctrl + shift + P -> TDT4102: Create Project from TDT4102 Template -> Inspera -> Inspera_2_Handout_Part_3

Den utdelte koden inneholder **IKKE** konfigurasjonsfilene som er nødvendig for å kjøre programmet. Du må derfor kjøre "TDT4102: Create Project from TDT4102 Template -> **Configuration only**" fra Comand Palette i VS Code. Dette skal gjøres i mappen du legger den utpakkede koden i, altså i mappen du skal jobbe med øvingen.

Når du vil levere, må du lagre filene, og levere dem som en zippet mappe i Inspera. Du kan bruke fagets VS Code Extension for å få til dette (cmd/ctrl + shift + P -> TDT4102: Prepare a zip file for delivery). **Husk at hvis du endrer på koden, må du først lagre og deretter lage zip-filen på nytt.**

PS: Når du løser en deloppgave kan du anta at alle de andre funksjonene du har laget fram til da fungerer som de skal, selv om dette ikke er tilfelle.



Figur 1: En MiniCraft-verden. Spilleren (figuren) skal kunne bevege seg rundt i verdenen, og plukke opp og plassere ut blokker. Spilleren i dette eksempelet har plukket opp 2 stein- og 4 gress-blokker.

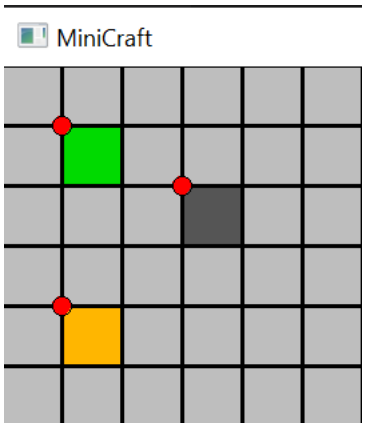
Intro:

I denne øvingen skal du utvikle spillet `MiniCraft`.

I `MiniCraft` skal man utforske og videreutvikle genererte "verdener". En verden består av av kvadratiske byggesteiner, også kalt blokker. I dette spillet finnes det to typer blokker, "stein" og "gress". Du kan se et eksempel på en MiniCraft-verden i figur 1. For å utforske verdenen skal man kunne styre en "spiller" med tastaturets piltaster. Man skal også kunne plukke opp blokker, og plassere dem andre steder i verdenen. Slik kan man bygge fine byggverk i verdenen.

For å gjøre oppgaven overkommelig, har du fått utdelt en del starter-kode. Denne vil bli presentert underveis, du trenger ikke sette deg inn i alt på en gang.

Lykke til!



Figur 2: Illustrasjon av rutenettet.

1 Block-klassen (16 poeng)

Oppgaven løses i *Block.cpp*

a) *6 poeng*
Implementer medlemsfunksjonen `Block::getBlockType`. Denne skal returnere blokktypen til et `Block`-objekt.

b) *10 poeng*
Implementer medlemsfunksjonen `Block::draw`, som tegner blokken som et kvadrat på riktig plass og med riktig farge i et vindu. Funksjonen tar inn et `AnimationWindow`, og et `Point` som representerer hvor det øvre venstre hjørnet av blokken skal plasseres på vinduet.

For å tegne et kvadrat kan man bruke funksjonen `AnimationWindow::draw_rectangle(Point topLeftPoint, int width, int height, Color color)`.

Farger: Parameteren `color` til `draw_rectangle` kan tas inn som et heltall som representerer fargen. For å sette riktig farge til riktig blokktype, kan man bruke tallverdiene til enumklassen `BlockType` som er deklarert i *Block.h*. Det vil si at en blokk av typen `Blocktype::Stone` skal tegnes med farge 8.

2 MiniCraftWindow-klassen. (0 poeng)

I denne oppgaven skal du sette deg inn i `MiniCraftWindow`-klassen. Oppgaven har ingen poeng, men er ment for å gjøre det lettere å forstå den utdelte koden. **Det er ikke meningen at du skal bruke mye tid her, du trenger ikke å se gjennom alt, men heller gå tilbake hvis det trengs.**

MiniCraft-verdenen er bygget opp av firkantede blokker. Det er derfor naturlig å beskrive den som et rutenett. Posisjonen i rutenettet defineres som avstanden i antall blokker fra det øvre venstre hjørnet, i henholdsvis x- og y-retning. Figur 2 illustrerer dette. Her har den grønne (øverste) blokken posisjonen (1,1), den grå (midterste) posisjonen (3,2) og den gule (nederste) posisjonen (1,4).

For å tegne rutenettet trengs også blokkenes vindukoordinater i piksler. Dette er hjørnet øverst til venstre, ettersom `AnimationWindow` tegner rektangler ut ifra dette hjørnet. I spillet er blokkstørrelsen på 30x30 piksler. Vindukoordinatene til blokkene i figur 2 er derfor henholdsvis (30,30), (90,60) og (30,120), markert med røde prikker.

Et vindukoordinat er **alltid** en `Point`-variabel, og en posisjon i rutenettet er **alltid** en `pair<int, int>`-variabel.

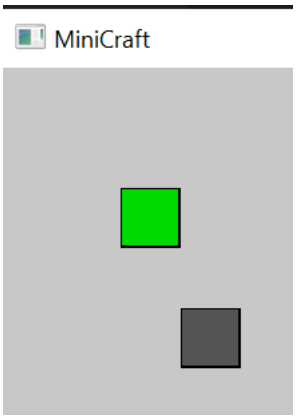
Medlemsvariabler :
Under er klassens medlemsvariabler. For å gjøre det mer oversiktlig, er variablene gruppert etter hvilken oppgave de er relevante fra.

- Relevant fra start:
  - `vector<shared_ptr<Block» blockGrid`
    - Inneholder rutenettet med blokker til spillet. Dersom en posisjon i rutenettet inneholder en blokk, er tilsvarende vektorelement en peker til blokken. Dersom posisjonen ikke er en blokk er vektorelementet `nullptr`.
    - Representerer et 2D rutenett, og skal derfor ha størrelsen Høyde · Lengde.
    - For å aksessere vektoren er det allerede definert flere medlemsfunksjoner. Disse nevnes under, og presenteres grundigere i seksjonen med medlemsfunksjoner.
      - `block_at_pos(...)` og `block_at_pt(...)` kan brukes til å hente eller endre blokker i vektoren.
      - `pos_at_pt(...)` kan brukes til å finne rutenett-posisjonen til et punkt i vinduet.
- Relevant fra oppgave 5:
  - `shared_ptr<Player> player`
    - En peker til spiller-objektet.
    - `nullptr` dersom vinduet ikke har en spiller.
  - `Point playerPoint`
    - Pikselposisjonen til det øvre venstre hjørnet av spilleren.
- Relevant fra oppgave 6:
  - `map <BlockType, int> inventory`
    - Et map med inventaret til spilleren.
  - GUI:
    - `Fl_Multiline_Output inventoryList`
    - `Fl_Choice markedInventoryType`
- Konstanter:
  - `blockSize`: Blokkstørrelsen i piksler
  - `gridHeight`: Høyden til rutenetter i antall blokker. Settes av konstruktøren.
  - `gridWidth`: Bredden til rutenettet i antall blokker. Settes av konstruktøren.
  - `playerHeight`: Høyden til spiller i piksler.
  - `playerWidth`: Bredden til spiller i piksler.
  - `widgetHeight`: Høyden til klassens widgets (knapper, tekstbokser) i piksler.
  - `widgetWidth`: Bredden til klassens widgets (knapper, tekstbokser) i piksler.

Medlemsfunksjoner :
Klassen har medlemsfunksjoner som allerede er implementert. Disse kan brukes til å løse oppgavene, og skal **ikke** endres med mindre noe annet er oppgitt. Oppgavene kan løses på ulike måter. Det er derfor ikke sikkert at du kommer til å bruke alle funksjonene.

I den utdelte koden finnes det også flere medlemsfunksjoner som brukes internt. Disse er det **ikke** nødvendig å hverken kalle på selv eller endre.

- Relevant fra start:
  - Konstruktøren:
    - Initialiserer AnimationWindow, GUI-elementer og konstante medlemsvariabler.
    - Du kan teste dine egne implementasjoner på den markerte plassen i konstruktøren.
  - `shared_ptr<Block> & block_at_pos(pair<int, int> pos)`
    - Returnerer blokk-pekeren til den aktuelle rutenett-posisjonen.
  - `shared_ptr<Block> & block_at_pt(Point pt)`
    - Returnerer blokk-pekeren til det aktuelle vindukoordinatet.
  - `pair<int, int> pos_at_pt(Point pt)`
    - Returnerer rutenett-posisjonen det aktuelle punktet på spillvinduet.
  - `void testBlockCreation();`
    - Funksjon for å teste om den grunnleggende implementasjonen er riktig. Videre beskrevet i oppgave 3*c*.
- Relevant fra oppgave 5:
  - `void updatePlayer();`
    - Kalles av den evige spilløkken `playMiniCraft()`. Du kan bruke denne funksjonen til å kalle funksjonene du implementerer i oppgave 5, slik at de kalles jevnlig i spillet.
  - `bool isPlayerOnGround(Point playerPos)`
    - Sjekker om en spiller med posisjonen `playerPos` har en blokk rett under seg.
    - Returnerer `true` dersom dette stemmer, og `false` hvis ikke.
  - `bool isPlayerPosLegal(Point playerPos)`
    - Sjekker om `playerPos` tilsvarer en lovlig spillerposisjon, altså ikke inne i en blokk eller utenfor verdenen.
    - Returnerer `true` dersom `playerPos` er lovlig, og `false` hvis ikke.
- Interne funksjoner:
  - `void initializeWorld()`
    - Initialiserer rutenettet til riktig størrelse.
    - Gjøres ved å lage `gridHeight` · `gridWidth` `nullptr`-ere i `blockGrid`.
  - `void playMiniCraft()`
    - Spillets evige løkke. Skal kalle medlemsfunksjonene `drawGrid()` og `updatePlayer()`, slik at brukerinput sjekkes jevnlig, og vinduet tegnes.
  - `bool inRange(Point pt)`
    - Skjekker om et punkt er innenfor spillvinduet.
  - `int handle(int event)`
    - En overlastet medlemsfunksjon til AnimationWindow, som håndterer brukerinput. Funksjonen kaller `getMouseInput()` ved museklikk på spillvinduet.
  - `void getMouseInput()`
    - Kalles av handle().
    - Sjekker hvilken knapp på musen som er trykket ned.
    - Kaller på funksjonene som skal implementeres i oppgave 6.



Figur 3: Forventet output fra testfunksjonen.

3 Tegn din første MiniCraft-verden. (37 poeng)

Oppgaven løses i *MiniCraftWindow.cpp*

I denne oppgaven skal du implementere den grunnleggende funksjonaliteten for å lage og tegne MiniCraft-verdener. For å initialisere verdenen til riktig størrelse, kalles medlemsfunksjonen `void createEmptyWorld()` i konstruktøren.

a) *6 poeng*
Implementer medlemsfunksjonen `Point pt_at_pos(pair<int, int> pos).`
Funksjonen skal ta inn en rutenettposisjon og returnere det tilsvarende vindukoordinatet.
Vindukoordinatet skal være 30 ganger større enn rutenettposisjonen. For eksempel tilsvarer:

- En rutenettposisjon på (2,3) vindukoordinatet (60,90).
- En rutenettposisjon på (0,4) vindukoordinatet (0,120).

b) *10 poeng*
Implementer medlemsfunksjonen
`void addBlock((pair<int, int> pos, BlockType type).`
Funksjonen skal lage en ny blokk i rutenettet med rutenettposisjon `pos` og blokktypen `type`. Dette skal gjøres ved å opprette en `shared_ptr<Block>` på det tilsvarende elementet i `blockGrid`-vektoren.
Du trenger ikke å ta hensyn til om rutenettposisjonen er tom fra før eller ikke.
Hint: Bruk den allerede definerte medlemsfunksjonen MiniCraftWindow::block_at_pos(...) for å hente ut riktig posisjon i rutenettet.

c) *0 poeng*
For å kunne teste mer avansert funksjonalitet, må det grunnleggende være riktig.
Test funksjonaliteten så langt ved å kalle den allerede definerte medlemsfunksjonen `void MiniCraftWindow::testBlockCreation()på den markerte plassen i konstruktøren`.
Funksjonen legger til to blokker av ulik type i rutenettet, og tegner disse i vinduet.
Du skal få opp grafikken i figur 3 hvis du har gjort riktig så langt.

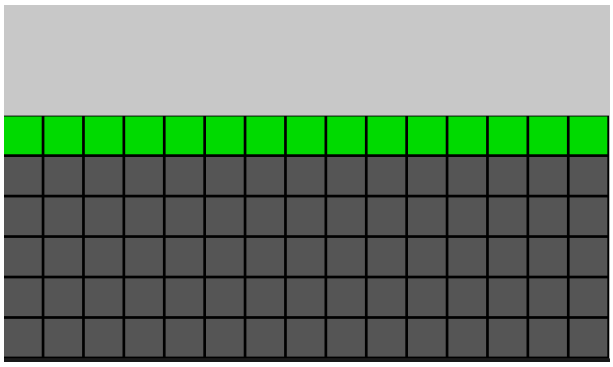
d) *15 poeng*
Implementer medlemsfunksjonen `void drawGrid().`
Funksjonen skal tegne alle blokkene i rutenettet, ved å tegne alle elementene i `blockGrid`-vektoren som ikke er `nullptr`.
Test funksjonen ved å legge til noen blokker i rutenettet i konstruktøren, og sjekk at de blir tegnet som forventet.
Tips: Du kan la deg inspirere av testfunksjonen fra forrige oppgave.

e) *6 poeng*

Implementer medlemsfunksjonen `void removeBlock(pair<int, int> pos).`

Funksjonen skal fjerne blokken på rutenettposisjonen `pos` ved å sette det tilsvarende elementet i `blockGrid`-vektoren til `nullptr`.

Du trenger ikke å ta hensyn til om plasseringen er tom fra før eller ikke.



Figur 4: En flat verden

4 Generer din første MiniCraft-verden. (37 poeng)

Oppgaven løses i *MiniCraftWindow.cpp*

En viktig del av MiniCraft er å kunne generere verdener. I denne oppgaven skal du generere ulike typer verdener.

a) *17 poeng*

Implementer medlemsfunksjonen `void generateFlatWorld(int surfaceHeight).`
Funksjonen skal generere en flat verden, ved å fylle opp de `surfaceHeight` laveste lagene i verdenen med blokker. Det øverste blokklaget skal ha gressblokker, og alle under skal ha steinblokker. For eksempel skal en `surfaceHeight`-verdi på "6" generere figur 4.

Du kan teste funksjonen din ved å legge inn et kall på den i konstruktøren til `MiniCraftWindow`.

MiniCraft hadde ikke vært så spennende med bare flate verdener. Vi ønsker derfor å kunne generere verdener med litt struktur.

b) *20 poeng*

Implementer medlemsfunksjonen `void generateSteepWorld().`
Funksjonen skal generere tilfeldige verdener med litt struktur.

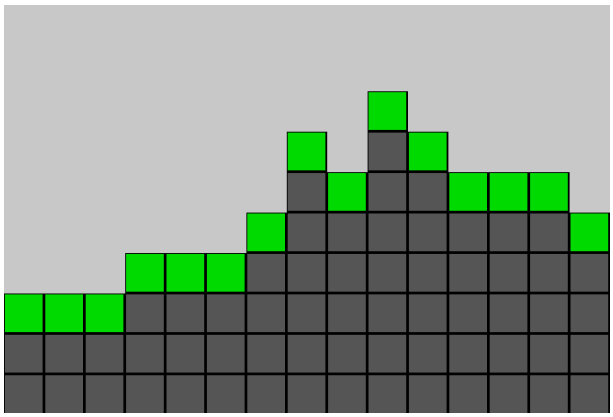
For at verdenen skal fremstå realistisk, innfører vi noen krav til hvordan den skal genereres.

- Ingen blokker skal flyte.
  - Alle blokker utenom det nederste laget skal ha en blokk under seg.
- Verdenen skal ikke være for bratt.
  - Hver kolonne i rutenettet skal ha en høydeforskjell på maks to blokker i forhold til de grensende kolonnene.
- Verdenen skal ha blokker i det nederste laget.
- Verdenen skal ikke ha blokker i de tre øverste lagene.
- Den øverste blokken i rutenettkolonnene skal være gress.
- Det skal genereres en ny, tilfeldig verden hver gang.

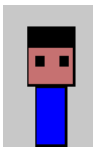
Ellers står du fritt til hvordan du vil løse oppgaven. Se figur 5 for hvordan en slik verden kan se ut.

Husk: Høyden og bredden til rutenettet er gitt i medlemsvariablene `gridHeight` og `gridWidth`.

Du kan teste funksjonen din ved å legge inn et kall på den i konstruktøren til `MiniCraftWindow`.



Figur 5: Et eksempel på en litt brattere verden. Merk at høydeforskjellen er på maks to blokker.



Figur 6: Spillerfiguren.

5 Styr din første MiniCraft-spiller (39 poeng)

Oppgaven løses i *MiniCraftWindow.cpp*

I MiniCraft skal man gå rundt som en spiller og utforske verdenen. Vi trenger derfor funksjonalitet for å ta inn brukerinput for å styre denne spilleren.
Spilleren representeres som et `Player`-objekt, og holdes av medlemsvariabelen `shared_ptr<Player> MiniCraftWindow::player`. Variabelen skal være `nullptr` hvis det ikke finnes en spiller. `Player`-klassen er definert i `Player.h`. Posisjonen til spilleren i spillvinduet lagres i medlemsvariabelen `Point MiniCraftWindow::playerPoint`. Posisjonen representerer hjørnet øverst til venstre hos spilleren.

For å styre spilleren må du sjekke brukerinput og tegne spilleren jevnlig. Dette kan du gjøre i medlemsfunksjonen `MiniCraftWindow::updatePlayer()`. Denne funksjonen kalles kontinuerlig av spilløkken.

Spilleren tegnes i MiniCraftWindow::updatePlayer(), med medlemsfunksjonen

`void Player::draw(AnimationWindow& win, Point upperLeftCorner)`.

For å starte spilløkken må du kalle `mw.playMiniCraft()` på den markerte plassen i `main()`.

a) *6 poeng*
Implementer medlemsfunksjonen `initializePlayer(Point playerPos)`.
Funksjonen skal lage spilleren, ved å initialisere

`shared_ptr<Player> MiniCraftWindow::player`.

Sett spillerposisjonen `playerPoint` slik at spilleren lages på angitt posisjon. Spilleren er to blokker høy, og tegnes ut ifra det øverste venstre hjørnet. Initialiser spilleren på den markerte plassen i konstruktøren til `MiniCraftWindow`.

Du skal få opp figur 6 på skjermen hvis du har gjort riktig.

Det kan være lurt å teste oppgaven med en flat verden, og initialisere verden før spiller.

NB! Figuren vil ikke tegnes hvis du ikke har startet spilløkken.

b) *10 poeng*
Implementer medlemsfunksjonenen `gravity()`.

Funksjonen skal flytte spillerposisjonen `playerPoint` 6 piksler nedover hvis spilleren ikke står på en blokk, dvs. ikke har en blokk direkte under seg.

Den allerede definerte medlemsfunksjonen `MiniCraftWindow::isPlayerOnGround(Point playerPos)` returnerer `true` dersom en spiller har en blokk direkte under seg, og `false` hvis ikke.

Kall funksjonen på den markerte plassen i medlemsfunksjonen `MiniCraftWindow::updatePlayer()`.

Du kan teste funksjonen ved å initialisere spilleren over bakken.

Nå skal vi styre spilleren med piltastene. En tastaturtast kan sjekkes med medlemsfunksjonen `AnimationWindow::is_key_down(KeyboardKey key)`. Funksjonen returnerer `true` dersom den aktuelle tasten er trykket ned, og `false` hvis ikke. For eksempel returnerer kallet `AnimationWindow::is_key_down(KeyboardKey::RIGHT)` `true` hvis den høyre piltasten er nede.

Du må også forhindre at spilleren går inn i blokker eller utenfor verdenen, altså ulovlige posisjoner. Lovligheten til en spillerplassering kan sjekkes med den allerede implementerte medlemsfunksjonen `MiniCraftWindow::isPlayerPosLegal(Point playerPos)`. Funksjonen returnerer `true` dersom input-variabelen `playerPos` tilsvarer en lovlig spillerposisjon, og `false` hvis ikke.

c) *8 poeng*

Implementer medlemsfunksjonenen `moveRight()`.

Funksjonen skal flytte spillerposisjonen `playerPoint` 2 piksler til høyre hvis den høyre piltasten `KeyboardKey::RIGHT` er nede og bevegelsen er lovlig.

d) *7 poeng*

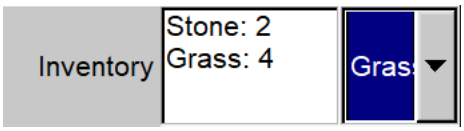
Implementer medlemsfunksjonenen `moveLeft()`.

Funksjonen skal flytte spillerposisjonen `playerPoint` 2 piksler til venstre hvis den venstre piltasten `KeyboardKey::LEFT` er nede og bevegelsen er lovlig.

e) *8 poeng*

Implementer medlemsfunksjonenen `jump()`.

Funksjonen skal flytte spillerposisjonen `playerPoint` 3 **blokklengder** oppover hvis oppoverpiltasten `KeyboardKey::UP` er nede, spilleren står på en blokk og bevegelsen er lovlig.



Figur 7: Vi har her fire gressblokker og to steinblokker i inventaret. Den markerte blokktypen er gress.

6 Spillerens inventar (51 poeng)

Oppgaven løses i *MiniCraftWindow.cpp*

I MiniCraft ønsker man å bygge og utvikle verdenen. For å endre på verdenen, må man kunne fjerne og legge til blokker. Når spilleren fjerner en blokk skal denne legges til i spillerens inventar. Inventaret skal holde oversikt over hvor mange av hver blokktype spilleren har, og representeres av medlemsvariabelen `map<BlockType, int> MiniCraftWindow::inventory`. Her spesifiserer nøkkelen en blokktype, og den tilhørende verdien antallet av blokktypen i inventaret.

Man skal kunne fjerne en blokk i griddet med et venstre museklikk, og legge til en blokk med høyre museklikk. Klikk registreres av den utdelte medlemsfunksjonen `MiniCraftWindow::handle(int event)`. Denne kaller videre medlemsfunksjonen `MiniCraftWindow::getMouseInput()`, som kaller medlemsfunksjonene `obtainBlock(...)` eller `placeBlock(...)` basert på klikktypen.

Funksjonene du skal implementere i denne oppgaven kalles derfor automatisk allerede.

Det er to GUI-objekter knyttet til inventaret: `Fl_Multiline_Output inventoryList` og `Fl_Choice markedInventoryType`. `inventoryList` skal vise hvilke og antallet blokker i inventaret.

`markedInventoryType` er en nedtrekksmeny som lar spilleren velge hvilken type blokk som skal plasseres.

a) *18 poeng*
Implementer medlemsfunksjonen `void obtainBlock(Point pt)` .
Funksjonen skal plukke opp en eventuell blokk på rutenettposisjonen tilsvarende vinduskoordinatet `pt`, og legge denne til i inventaret. Dette betyr:

- Sjekke om det finnes en blokk i `pt`. Hvis ja:
  - Fjerne blokken fra rutenettet.
  - Inkrementere blokkens type i inventaret med én.
  - Hvis blokktypen ikke fantes i inventaret fra før, legge denne blokktypen til i valgmenyen `markedInventoryType`

For å legge til nye valg i en `FL_Choice` kan du bruke medlemsfunksjonen `add()`.
Tips: Noen funksjoner i grafikk-biblioteket krever at input er en `const char *`. Dersom du har tekststrengen din i en variabel må du da skrive `yourStringName.c_str()` som input til grafikk-funksjonen (og ikke bare `yourStringName`).
Du kan få bruk for den eksisterende medlemsfunksjonen `shared_ptr<Block> block_at_point(Point pt)`.

b) *15 poeng*
Implementer medlemsfunksjonen `updateInventoryList()`.
Funksjonen skal oppdatere listen med inventarets innhold som vises til høyre på skjermen, `Fl_Multiline_Output::inventoryList`. Se figur 7 for et eksempel. Innholdet i inventoryList oppdateres med `inventoryList.value(string val)`, og hver linje skal ha formatet "[Type blokk]: [Antall]". Funksjonen kalles allerede i `getMouseInput()`.

c) *18 poeng*
Implementer medlemsfunksjonen `placeBlock(Point pt, BlockType type)`.
Funksjonen skal plassere en blokk på rutenettposisjonen tilsvarende vindukoordinatet `pt` dersom den er tom. Dette betyr:

- Sjekke om det finnes en blokk i `pt`. Hvis nei:
  - Sjekke hvilken blokktype som er markert i valgmenyen `markedInventoryType`.
    - Hvis ingen blokktype er markert, skal funksjonen terminere.
  - Legge til en blokk av den markerte blokktypen i den tilsvarende rutenettposisjonen.
  - Dekrementere den markerte blokktypen i inventaret med én.
  - Sjekke om man tok den siste blokken av blokktypen fra inventaret. Hvis ja:
    - Fjerne den aktuelle blokktypen, altså nøkkelparet i map-et, fra inventaret.
    - Fjerne den aktuelle blokktypen fra valgmenyen

For å sjekke hva som er markert i valgmenyen, må du først finne indeksen med medlemsfunksjonen `Fl_Choice::value()`. Blokktypen kan du videre finne med medlemsfunksjonen `Fl_Choice::text(int index)`. Du kan fjerne et menyvalg med medlemsfunksjonen `Fl_Choice::remove(int index)`. Du må kalle på medlemsfunksjonen `Fl_Choice::show()` for å vise endringer. Se eksempelet under:

```
int idx = markedInventoryType.value();
string choice = markedInventoryType.text(idx);
markedInventoryType.remove(idx);
markedInventoryType.show();
```