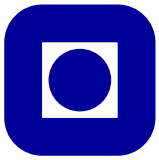


Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2023

Øving 4

Frist: 2023-02-10

Mål for denne øvingen:

- Lære å bruke referanser
- Jobbe med `string`, funksjoner og `struct`
- Lage et fullstendig program (et enkelt spill)
- Kode grafikk

Generelle krav:

- Alle funksjoner som defineres i `test.cpp` skal kalles fra main().
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Bruk Visual Studio Code, med mindre du kjenner et annet utviklermiljø bedre.

Anbefalt lesestoff:

- Appendix B.8.1 og B.8.2 i læreboka
- §4.3.1 i læreboka
- §8.5.3 - 8.5.6 i læreboka

1 Pass-by-value vs. pass-by-reference (15%)

Nyttig å vite: Pass-by-reference

De fleste funksjonene vi har sett på så langt i øvingsopplegget har tatt inn argumenter på såkalt «pass-by-value»-form. «Pass-by-value» vil si at verdien vi får inn som argument er en kopi av originalen. Hvis vi endrer på den, vil ikke endringen gjenspeiles i originalverdien som ble sendt inn. «Pass-by-reference» derimot, oppretter et alias til objektet som befinner seg utenfor funksjonen. Når det utføres en operasjon på referansen utføres det en operasjon på det opprinnelige objektet.
For å ta inn et argument som referanse må du bruke &. F.eks.

```
int incrementByValueNumTimes(int& startValue, int increment, int numTimes);
```

Her blir `startValue` tatt inn som en referanse.
Se §8.5.4 til 8.5.6 i læreboka for illustrasjoner og argumentasjon for når verdi og (const) referanse bør benyttes.

a) Kodeforståelse:
Når programmet under har kjørt ferdig, hvilken verdi er skrevet ut for `v0`?

```
int incrementByValueNumTimes(int startValue, int increment, int numTimes) {
    for (int i = 0; i < numTimes; i++) {
        startValue += increment;
    }
    return startValue;
}

void testCallByValue() {
    int v0 = 5;
    int increment = 2;
    int iterations = 10;
    int result = incrementByValueNumTimes(v0, increment, iterations);
    cout << "v0: " << v0
         << " increment: " << increment
         << " iterations: " << iterations
         << " result: " << result << endl;
}

int main() {
    testCallByValue();
}
```

b) utilities
Opprett en ny fil `utilities.cpp` med tilhørende headerfil. Legg til funksjonen `incrementByValueNumTimes()` fra forrige deloppgave i filen.

c) Tester
Opprett en ny fil `tests.cpp` med tilhørende headerfil, og legg til funksjonen `testCallByValue()` i `tests.cpp`. Denne funksjonen skal teste at `incrementByValueNumTimes()` virker. Kall `testCallByValue()` fra `main()`, gjerne med testmeny som i øving 2. `main()` skal være i `main.cpp`.

d) Funksjoner med referanseparameter
Lag en ny funksjon `incrementByValueNumTimesRef()`, som gjør akkurat det samme som `incrementByValueNumTimes()`, men bruker referanse. Funksjonen skal ikke returnere noe. Lag en ny funksjon `testCallByReference()` som tester

`incrementByValueNumTimesRef()` på tilsvarende måte som `testCallByValue()` tester `incrementByValueNumTimes()`. *Husk å oppdatere headerfilene.*

e) Swap

Skriv en funksjon `swapNumbers()` som bytter om på to heltallvariablers verdi. Funksjonen skal ligge i `utilities.cpp`. Bør denne funksjonen bruke referanser? Begrunn svaret ditt. Test gjerne med både pass-by-value, pass-by-reference og pass-by-const-reference og se hva som skjer.

2 Struct (15%)

Nyttig å vite: `struct`

En `struct` er en brukerdefinert type. I en `struct` kan man samle variabler av forskjellige typer. Disse variablene kalles medlemsvariabler. `struct`er brukes til å representere ting som ikke kan representeres med en type som allerede er definert. For eksempel kan man bruke en `struct` til å representere en film.

```
struct Movie{
    string title;
    int releaseYear;
};
```

Det er to måter å opprette og initialisere en `struct`-variabel. Den første måten deklarerer man først variabelen, og så initialiserer man medlemsvariablene med dot-operatoren.

```
Movie matrix;
matrix.title = "The Matrix;
matrix.releaseYear = 1999;
```

Med den andre måten gjør man begge deler på en gang.

```
Movie matrixTwo {"The Matrix Reloaded", 2003};
```

For å få tilgang på verdiene til medlemsvariablene kan man bruke dot-operatoren.

```
string movieTitle = matrix.title;
```

En av fordelene med å bruke sammensatte datatyper er at man slipper å holde styr på de individuelle delene av datatypen manuelt. Programmereren trenger ikke å passe på hvilke strenger og heltall som hører sammen til hvilken film, de henger alltid sammen siden de er samlet i en egen film-datatype.

Dersom man skal lage en funksjon som skal returnere releaseYear fra en Movie struct kan man gjøre det på følgende måte:

```
int getReleaseYear(Movie mov) {
    return mov.releaseYear;
}
```

a) Lag structen `Student` i `utilities.h`

`Student` skal holde variablene:

- name, av typen `string`
- studyProgram, av typen `string`
- age, av typen `int`

b) Definer funksjonen `printStudent()`

Funksjonen skal defineres i `utilities.cpp`. Funksjonen skal ta inn en `Student` som parameter, og skrive en *pen* utskrift av studentens `name`, `studyProgram` og `age` til skjerm. Du velger selv formatet på utskriften.

c) Definer funksjonen `isInProgram()`. Funksjonen skal defineres i `utilities.cpp`. Funksjonen skal ta inn en `Student` og en `string` som parameter og returnere hvorvidt studenten går på studieprogrammmet angitt av strengparameteren.

d) Test structen og funksjonene fra `main()` Det forventes at du alltid tester koden din, selv om det ikke er oppgitt i oppgaveteksten. Hvis du er usikker på hvordan du kan teste koden din kan du spørre en læringsassistent.

e) Oppgave om feilmeldinger

Anta at vi har laget egne filer, `tests.h` og `tests.cpp`, for å teste `Student`-structen, slik at `main.cpp`, `utilities.h` og `tests.h` ser ut som følger.

Hovedfil(main.cpp)

```cpp
#include "std_lib_facilities.h"
#include "utilities.h"
#include "tests.h"
int main(){
    testStudentStruct();
    return 0;
}
```

Headerfil(utilities.h)

```cpp
struct Student{
    // * * *
};
```

Headerfil(tests.h)

```cpp
#include "utilities.h"
void testStudentStruct();
```

Anta videre at man får følgende feilmelding i terminalen når koden kjøres:

```
error: redefinition of 'Student'
```

Forklar hva som skyldes denne feilmeldingen.

Hint: Feilen er knyttet til inkludering av h-filer.

3 `string` og behandling av tegn (char) (20%)

Nyttig å vite: "char-aritmetikk"

En bokstav, `char`, er i bunn og grunn en tallverdi (`'\0'` = 0), og dermed kan vi bruke regneoperasjoner:

```
// finner den n-te bokstaven i alfabetet (A-Z)
char c = 'A' + n - 1;
```

Samsvaret mellom tegn og tall finner du i en ASCII-tabell.

a) Definer funksjonen `testString()`

Defineres i `tests.cpp`. Funksjonen har ingen parameter og skal ikke returnere noe. Denne funksjonen vil bli bygd opp i løpet av resten av oppgave 3.

b) Definer funksjonen `randomizeString()`

Defineres i `utilities.cpp`. Funksjonen skal returnere en `string` av en gitt lengde som inneholder tegn i et gitt intervall. Parametere er antall tegn strengen skal bestå av, samt en øvre og nedre grense for tegn strengen kan fylles med (f.eks. `'A'` og `'G'`). Bruk det du allerede har lært i øving 3 for å generere tilfeldige tall.

c) Tilfeldige karakterer

Opprett en `string grades` i `testString()`. Stringen `grades` skal inneholde 8 tilfeldige karakterer, altså tegn fra A til og med F. Skriv `grades`-stringen til skjerm inne i `testString()`.

d) Definer funksjonen `readInputToString()`

Defineres i `utilities.cpp`. Funksjonen skal returnere en `string` av lengde `n`. Der tegnene i strengen skal ligge innenfor en øvre og nedre grense. Parametere er en øvre og nedre grense for hvilke tegn som er tillatt og `n`. Strengen som returneres skal fylles vha. konsollinput, `cin`. Hvis input ikke ligger innenfor nedre og øvre grense skal funksjonen be brukeren om ny input, her bryr vi oss ikke om input er store eller små bokstaver, slik at begge deler skal godkjennes. ("aA" == "Aa"). For videre bruk av funksjonen vil det være lurt å returnere strengen som enten bare store eller bare små bokstaver.

Se Appendix B.8.1 for funksjoner som kan hjelpe deg med å konvertere mellom store og små bokstaver, finne ut om det du leser er en bokstav, siffer, e.l.

e) Definer funksjonen `countChar()`

Defineres i `utilities.cpp`. Funksjonen tar inn en `string` og en `char`. Funksjonen skal returnere antall forekomster av tegnet i strengen som er sendt til funksjonen.

f) Telle karakterer og gjennomsnitt.

Opprett en `vector` med heltall, `gradeCount`, i `testString()`. Den skal romme antall forekomster av hver karakter (A-F).

Bruk `countChar()` til å fylle `gradeCount` med antallet forekomster av hver karakter i `grades`. Regn ut og skriv snittkarakteren til konsollen. (La A tilsvare 5, B tilsvare 4, osv, slik at snittet kan beskrives med tall).

4 Mastermind (25%)

I denne deloppgaven skal du implementere spillet Mastermind. Programmet ditt skal lage en tilfeldig kode på 4 bokstaver bestående av tegnene i intervallet [A, F]. Brukeren av programmet skal deretter gjette hvilke bokstaver koden inneholder, og hvilken rekkefølge de er i. Etter hver gang brukeren har gjettet, skal programmet fortelle brukeren hvor mange bokstaver brukeren gjettet riktig og hvor mange bokstaver som ble riktig plassert. Det er ikke nødvendig å implementere programmet ditt nøyaktig som beskrevet under, så lenge funksjonaliteten blir den samme og du gjenbruker kode fra tidligere i øvingen.

a) Grunnleggende oppsett.

Lag en ny fil som inneholder funksjonen som starter spillet. La funksjonen hete `playMastermind()`. Kall den fra `main`. Definer følgende heltallskonstanter (`constexpr`) i `playMastermind()`:

- `size`, antall tegn i koden, 4.
- `letters`, antall forskjellige bokstaver, 6.

Hvorfor bruker vi `constexpr` og ikke `const` her?
Når kunne det være hensiktsmessig å benytte `const` framfor `constexpr` for dette spillet?
Du kan lese om `constexpr` og `const` i §4.3.1 i læreboken.

b) Datastruktur. Lag to tekststrenger (`string`) i `playMasterMind()`

- `code` - Skal inneholde koden spilleren skal prøve å gjette.
- `guess` - Skal inneholde bokstavene spilleren gjetter.

c) Generer kode.

Bruk funksjonen `randomizeString()` til å fylle `code` med `size` tilfeldige bokstaver mellom `'A'` og `'A'` + (`letters` - 1). Ved å bruke verdiene fra a) skal `code` inneholde fire bokstaver fra `'A'` til og med `'F'`.

d) Spillerinput.

Bruk `readInputToString()` til å spørre spilleren etter `size` antall bokstaver, og lagre dem i `guess`.

e) Korrekt plassering.

Skriv funksjonen `checkCharactersAndPosition()`, som skal returnere hvor mange riktige bokstaver spilleren har på riktig posisjon. Svaret skal returneres som et heltall.

f) Korrekt bokstav, uavhengig av posisjon.

Skriv funksjonen `checkCharacters()`, som skal returnere hvor mange riktige bokstaver spilleren har gjettet, uavhengig av posisjon. Resultatet skal være hvor mange bokstaver som befinner seg i både kode og gjett, om de er på samme posisjon i kode og gjett er likegyldig. Svaret skal returneres som et heltall.

Eksempel: koden er ABCD. Om man gjetter CDEE vil C og D være bokstaver som er en del av koden og denne funksjonen skal returnere 2. En annen gjetning kan være BACD, alle bokstavene er med i koden, men kun to er på korrekt plass. For BACD skal funksjonen likevel returnere 4, for den bryr seg kun om hvor mange av bokstavene som er med i både koden og gjettet.

HINT: Iterer gjennom mulige bokstaver i kodene, i stedet for å iterere gjennom en av kodene bokstav for bokstav. Det kan også være lurt å bruke `countChar()`.

g) Spill-løkke.

Utvid koden fra **d)** slik at programmet spør spilleren etter en ny kode så lenge `checkCharactersAndPosition()` returnerer et tall mindre enn `size`.

h) Fullfør spillet.

Flett sammen spillogikken i `playMastermind()`. Når du er ferdig skal programmet lage og lagre en tilfeldig kode som spilleren kan gjette på inntil den rette koden er funnet. For hver kode spilleren gjetter skal programmet skrive ut hvor mange riktige bokstaver spilleren gjettet, og hvor mange av dem som var på rett plass.

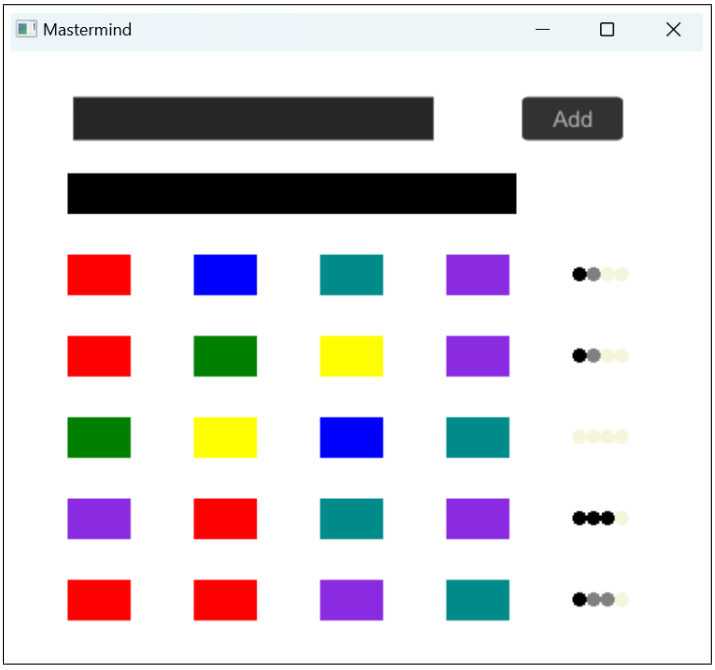
i) Begrenset antall forsøk.

Utvid koden din slik at spilleren har et begrenset antall forsøk på å gjette koden.

j) Seier eller tap.

Utvid spillet til å gratulere spilleren med seieren (eller trøste den etter tapet).

5 Mastermind med grafikk (25%)



Figuren over viser en grafisk framstilling av spillet du har programmert. Den øverste raden med rektangler i vinduet representerer den hemmelige koden som skal gjettes. Hvert rektangel representer en bokstav. De resterende radene med rektangler er gjetninger brukeren har utført. Sirklene til høyre for rektanglene gir brukeren tilbakemelding om hvor mange bokstaver de har gjettet riktig, og hvor mange bokstaver som er gjettet i korrekt posisjon.

Utdelt kode, `masterVisual.cpp` og `masterVisual.h` skal brukes til å gi en grafisk framstilling av spillet. Det er ikke nødvendig å forstå den utdelte koden for å løse oppgaven. Det du skal bruke av den utdelte koden blir forklart. På slutten av kurset vil du være i stand til å forstå den utdelte koden.

Den utdelte koden kan hentes på samme måte som beskrevet i siste oppgave i øving 3.

Skalering

Når man jobber med grafikk er det lurt å bruke konstanter for størrelsene på elementene i vinduet. I denne oppgaven skal du bruke en veldig enkel skalering, der størrelsen og plasseringen av rektanglene og sirklene er avhengig av hvor stort vinduet er, og hvor mange elementer det er i vinduet.

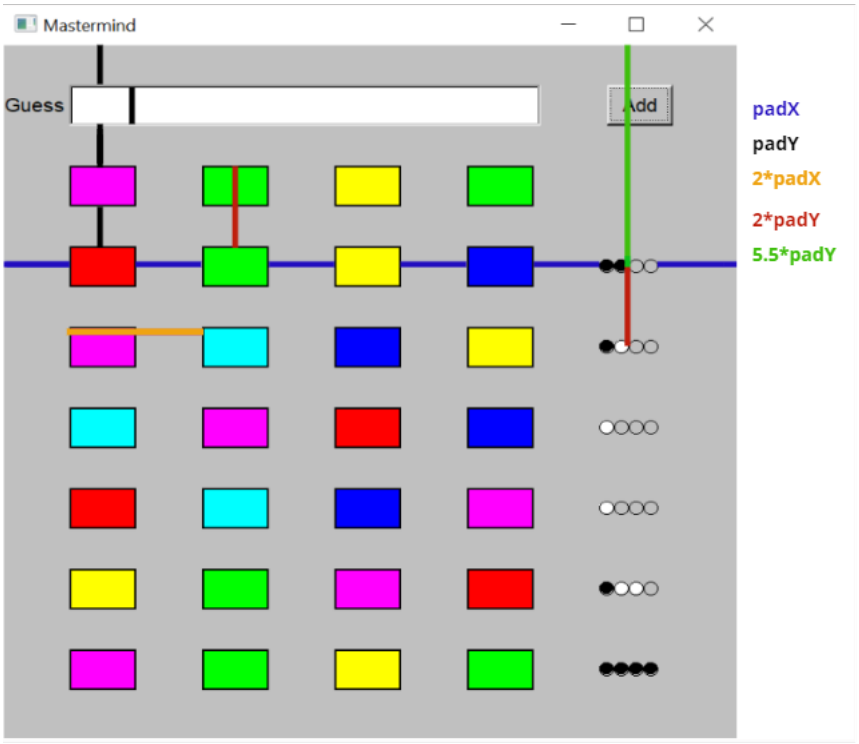
I `masterVisual.h` er det definert en del konstanter. Konstantene du skal endre er `winW, winH, padX, padY` og `radCircle`.

`winW` og `winH` er vinduets bredde og høyde.

`padX` er avstanden mellom hvert element i x-retning og bredden til hvert rektangel og bredden til alle sirklene i x-retning. Med element menes det her et rektangel eller alle sirklene i x-retning. `padX` er vinduets bredde delt på to ganger antall elementer i x-retning pluss en. F.eks. hvis du skal gjette en kode på fire bokstaver: `padX=winW/(2*5+1)`. Siden du har fem elementer i x-retning, fire rektangler og ett element med sirklene.

`padY` er avstanden mellom hvert element i y-retning og høyden til hvert rektangel. `padY` er vinduets bredde delt på antall elementer i y-retning ganger to pluss en. F.eks. hvis du skal spille seks runder er `padY=winH/(2*8+1)`. Siden du har seks elementer til de seks rundene i tillegg har du to elementer i tekstboksen og koden som skal gjettes.

`radCircle` er radiusen til hver av sirklene. Siden alle sirklene til sammen skal ha bredde `padX` så er radiusen til hver sirkel `padX` delt på antall sirkler ganger to. F.eks. hvis du skal gjette fire bokstaver er `radCircle=padX/8`. Resten av konstantene bestemmer plasseringen til add-knappen og tekstboksen. Figuren nedenfor viser Mastermindvinduet med mål.



a) Grafikkvindu

Bestem hvor mange bokstaver som skal gjettes, og hvor mange runder som skal spilles. Vi anbefaler fire bokstaver og seks runder. Bestem i tillegg størrelsen på vinduet ved å endre verdiene til `winW` og `winH` i `masterVisual.h`. Vi foreslår en bredde på 500 og høyde på 500 piksler. Deretter definer `padX, padY` og `radCircle` i `masterVisual.h`.

Opprett en ny funksjon med navn `playMastermindVisual()` i `masterVisual.cpp` med samme innhold som `playMastermind()` (Copy-Paste).

I denne øvingen skal du benytte datatypen `MastermindWindow` som er definert i

`masterVisual.h`. Opprett vinduet (som du kaller `mwin`) ved å plassere følgende kodelinje i `playMastermindVisual()`:

```
MastermindWindow mwin{800, 20, winW, winH, size, "Mastermind"};
```

Koordinatene 900, 20 angir posisjonen til øvre venstre hjørne av vinduet. `winW` og `winH` er bredden og høyden, `size` er antall tegn i koden, og det siste argumentet er tittelen til vinduet.

I denne øvingen bør du opprette *ett nytt vindu for hvert spill*, da vil programmet rydde opp etter seg selv og hvert nye spill starter med et tomt vindu.

Du må i tillegg bytte ut funksjonen `readInputToString()` med `mwin.getInput()` i `playMastermindVisual()`. `getInput()` leser inn tekst fra tekstboksen i Mastermindvinduet. Den sjekker at gjetningen er på riktig format, og bare inneholder lovlige tegn. Funksjonen returnerer en `string` med store bokstaver.

b) Visning av kode og gjetning

Din oppgave er nå å definere funksjoner som skal tegne en rad med fargede rektangler i vinduet basert på en gjetning. Vi benytter `map`-et `colorConverter` for å konvertere tallverdier til farger. En `int` konverteres til en `Color` som dette: `colorConverter.at(intName)`. Tallverdien 1 = rød, 2 = grønn, 3 = gul, 4 = blå, 5 = lilla, 6 = cyan. Først skal funskjonen `addGuess()` defineres. Den skal legge til et `Guess` i vektoren `guesses`. `Guess` er en struct som holder variablene `code` og `startLetter`. Parameterne til funksjonen er en referanse til vinduet du har opprettet, bokstavkoden som er gjettet og første bokstav som kan gjettes.

Deretter skal denne vektoren brukes til å tegne raden med rektangler for hver runde i funksjonen `wait_for_guess()`. I `wait_for_guess()` står det allerede skrevet noe kode, men selve tegningen av rektanglene for hvert gjett mangler. Et rektangel tegnes i vinduet ved å skrive:

```
draw_rectangle(Point{xPos, yPos}, padX, padY, c);
```

Her er `Point{xPos,yPos}` posisjonen til hjørnet oppe til venstre av rektangelet, `padX` og `padY` er bredden og høyden til rektanglet. `c` er en farge.

Husk at `yPos` er avhengig av runde-nr, mens `xPos` starter på samme sted hver runde. Du må inkrementere `xPos` for hvert rektangel du tegner. Fargen `c` er avhengig av hvilken bokstav som er gjettet.

Ved å gjøre dette vil man som i eksempelfiguren få en rad med rektangler for hver runde.

Ved utvikling og debugging av programmet kan det være nyttig å spille med synlig kode. Det kan du oppnå ved å kalle `addGuess()` med den hemmelig bokstavkoden som den andre parameteren og skrive følgende kodelinje:

```
mwin.setCodeHidden(false);
```

Du må plassere kall på `addGuess()` på egnet sted inne i funksjonen `playMastermindVisual()`.

Når du mener logikken i programmet ditt er helt riktig kan du endre det til å spille med skjult kode. Det gjøres ved å endre variabelen i setCodeHidden fra over til `true` istedet for `false`. Det vil da bli tegnet et sort rektangel oppå den øverste raden som tidligere viste den hemmelige koden.

c) Vis antall korrekte plasseringer

I denne deloppgaven skal du kode tilbakemeldingsdelen av mastermind-spillet, dvs. de sorte og hvite sirklene i høyre del av figuren. `addFeedback()` er en funksjon som skal legge til en `Feedback` i vektoren `feedbacks`. `Feedback` er en struct som holder variablene `correctPosition` og `correctCharacter`. Parameterne er vinduet, antall korrekte plasseringer og antall korrekte bokstaver med feil posisjon.

Videre skal `feedbacks`-vektoren brukes til å tegne sirkelene i `wait_for_guess()`. Man tegner en sirkel ved å skrive:

```
draw_circle(Point{xPos, yPos}, radCircle, indicatorColour);
```

`Point{xPos,yPos}` er her sentrum av sirkelen og `radCircle` er radiusen. Bruk sort sirkel for å markere korrekt bokstav med korrekt posisjon og hvit for korrekt bokstav uavhengig av posisjon.

Her kan det være gunstig å bruke datatypen `Color` for å få riktige farger, dersom man vil sirkelen skal være rød kan man gjøre det på følgende måte:

```
Color indicatorColour = Color::red;
```

Plasser kallet av `addFeedback()` på egnet sted i `playMastermindVisual()`.