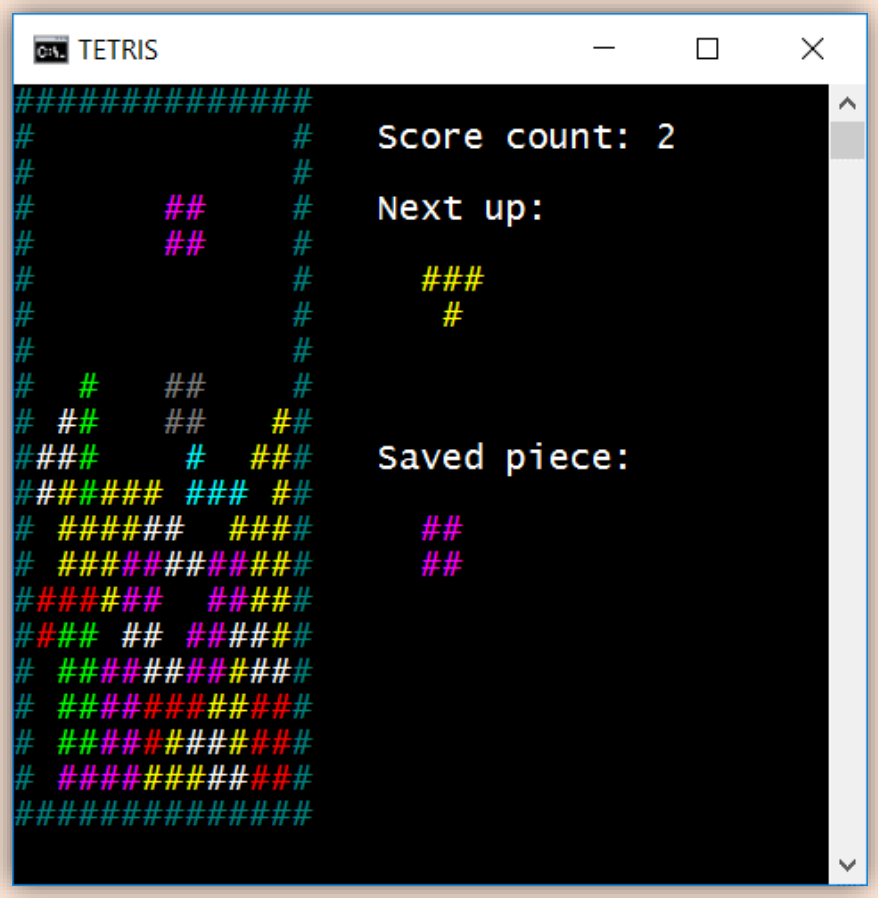
COMPUTERSPIL

Af: Carl Schou



INDHOLDSFORTEGNELSE

INDLEDNING

Mit spil er en form for moderne tetris, der i modsætning til andre offentlige, open-source tetris-spil er baseret kun på tekst. Spillet er specifikt lavet til Windows' standardkonsol (Win32 konsol via conhost[1]) og understøtter derfor ikke andre operativsystemer (men grundet konsol- brugeroverfladen interne struktur, ville det ikke være krævende at omskrive til andre operativsystemer). Jeg har valgt at lave spillet i moderne C++ (STD version 17[2], seneste version) da det er fleksibelt og meget low-level, hvilket tillader mig at optimere spillet meget mere end hvis jeg arbejdede med et high-level sprog som for eksempel Python. Alt i projektet er lavet helt fra bunden, inkluderet konsol-brugeroverfladen der er fungerer via en form for buffer logik. Koden er skrevet og dokumenteret i engelsk, da det føles naturligt for mig, det er noget jeg gør i alle programmeringsprojekter. Det er også en fordel at skrive alting på engelsk, hvis man vil publicere sit arbejde, på for eksempel GitHub[3], når det er færdigt, da det ikke er optimalt at udgive ikke-engelsk arbejde globalt.

[1] https://en.wikipedia.org/wiki/Win32_console
[2] https://en.wikipedia.org/wiki/C++17
[3] https://github.com/

VALG AF PROGRAMMERINGSSPROG

Jeg foretrækker personligt C++ over alle andre sprog, da jeg ikke mener, at high-level/scripting sprog burde bruges til større projekter hvor hastighed er en primær faktor indenfor brugervenlighed og programeffektivitet. Jeg bruger personligt moderne C++ til alle mine projekter, da jeg godt kan lide at være fuldstændig i kontrol både med variabler i hukommelse og selve maskinkode outputtet, da det er svært at optimere et programmer i et high-level sprog. At lave computerspillet i C++ har også en masse fordele:

1. Ved at sproget er et low-level programmerings sprog, er det muligt at arbejde med Windows' api (WinApi[4]) meget nemmere end hvis jeg brugte et programmeringssprog som Python, C# eller Ruby. Man kan selvfølgelig godt køre WinApi funktioner fra high-level sprog via interopability, C# har for eksempel System.Runtime.InteropServices[5] der gør det muligt at importere funktioner fra Windows libraries som en delegate[6], men det kan være et problem da high-level programmerings sprog sjældent er konsekvente i deres rå definitioner i hukommelsen af forskellige objekt typer (int, string med mere), strings i C# er for eksempel helt anderledes i hukommelses end standard null-terminatedede[7] strings, hvilket ville forsage 'undefined-behaviour'[8] grundet, at de ikke er afsluttet med en null karakter (0x00)

[4] https://en.wikipedia.org/wiki/Windows_API
[5] https://msdn.microsoft.com/en-us/library/system.runtime.interopservices(v=vs.110).aspx
[6] En form for dynamisk funktion på .NET platformen
[7] Null-terminated betyder at datastrukturen er afsluttet af en 'null' karakter (hex 0x00), dette gør, at man ved hvornår dataen slutter.
[8] Undefined-behaviour (UB) er et udtryk der bruges, når resultatet af en operation ikke er direkte angivet, dette kan for eksempel være ved direkte manipulation eller korruption af hukommelse.

2. Takket være moderne compilere tilbyder C++ en masse forskellige optimeringsfordele der gør det egnet at bruge i høj-præstations programmer. Jeg snakker lidt mere om hvordan jeg har valgt at optimere spillet senere i stykket **'Optimeringer'**. High-level sprog er begrænset af deres JIT[9], hvilket sjældent producerer den hurtigste version af en funktion og ville skulle bruge tid på at compile selve programmet i mens det bliver kørt, hvilket er langt fra optimalt når man har brug for præstation. Dette betyder ikke high-level sprog er langsomme eller, at de ikke kan være hurtige, det er bare ikke et konsekvent resultat og derfor bliver det sjældent brugt i præstationspåvirkede programmer med mere. Der er selvfølgelig nøglehuller og tricks til at få high-level sprog til at blive hurtigere, man kan i C# for eksempel få mscoree.dll[10] til at compile hele ens program på en gang for at undgå præstationsnedsættelsen der forsages af JIT compilere. C# bruges trods alt i moderne game-engines som for eksempel Unity[11], så det er ikke umuligt at få det til at køre hurtigt.
3. Moderne C++ (revision 17) tilbyder også en masse fantastiske egenskaber der gør at jeg foretrækker dette over alle andre programmeringssprog. De små ting som structured-bindings[12] , templates [13]og automatisk variable-type deklaration [14]gør det en fornøjelse at arbejde med dette fleksible programmeringssprog. Det fede ved C++ er, at man altid ved hvordan ting virker, da man ikke er afhængig af en tredjepart som en JIT compiler. Hvis programmet så ikke virker, er det lige ud af landevejen at debugge [15] programmer ved at gå i gennem maskinkoden, for maskinkode lyver ikke.

[9] Just-In-Time, et udtryk der bruges til at beskrive high-level programmerings- og scriptingsprogs compiler. Navnet kommer fra metoden disse high-level compilere fungerer, da de først compiler når koden skal bruges, altså 'Just-In-Time', dette kan forsage præstationsproblemer. Modsætningen for Just-In-Time kaldes Ahead-Of-Time
[10] Microsoft .NET's JIT compiler
[11] https://unity3d.com/
[12] http://en.cppreference.com/w/cpp/language/structured_binding
[13] http://en.cppreference.com/w/cpp/language/templates
[14] http://en.cppreference.com/w/cpp/language/auto
[15] https://en.wikipedia.org/wiki/Debugging

BRAINSTORM

Mit første indtryk var at lave et computerspil der var konsol-baseret, hvilket også var den basis jeg i sidste ende valgte at bruge. Det strejfede selvfølgelig min hjerne at arbejde med en grafisk brugeroverflade som direct3d, men jeg kunne lide tanken om udfordringen konsol-baserede spil giver, da det krævede at jeg lavede mit eget grafik-system for at tegne de henholdsvise figurer. Min første tanke var at lave en form for telltale [16]rollespil, men det var åbenbart en populær tanke i klassen og det virkede heller ikke udfordrende nok. Jeg overvejede også at lave en form for overlevelses-spil, men det ville være for kompliceret at tegne figurerne i Windows konsollen. Til sidst kom jeg på ideen om at lave tetris eller sænker slagskib, og det blev så Tetris der vandt kampen. Tetris er også en solid idé, da det er en spil-form som mange kender og den er enkelt at gå til for brugeren. Der er ikke mange usikkerheder omkring tetris spil, og spillets regler er klart defineret i Tetris standarden[17].

[16] En form for RPG (roleplaying-game) hvor ens beslutninger påvirker spillet.
[17] http://tetris.wikia.com/wiki/Tetris_Guideline

KRAV

KRAVSPECIFIKATION

Mit spil skal udfylde de henholdsvise krav, for både at udfylde tetris standarden og for at tilfredsstille mig. Jeg har valgt disse krav da jeg føler, at de er essentielle og er specifikke nok til at få et reelt produkt der ligner min prototype ud af det. Det er vigtigt at kravene ikke er brede og åbne for misfortolkelse af producenten eller programmøren.

1. Visuelle Krav
    a. Brikkerne skal tegnes med hashtags
    b. Skal være konsol-baseret
    c. Baggrunden skal være sort, og brikkerne skal have forskellige farvers
2. Informationsmæssige krav
    a. Den skal vise næste tetris stykke
    b. Den skal vise score
    c. Vis hvor bevægende tetris stykke vil lande
3. Interaktionsmæssige krav
    a. Styres med piletasterne
    b. Mellemrum skal rykke brikkerne ned
    c. 'C' tasten skal gemme nuværende stykke

TESTSPECIFIKATION

Det antages at spilleren er kompetent nok til at bruge en computer, og kompetent nok til at vide hvordan man bruger operativsystemet Windows. Det er også vigtigt, at brugeren der tester kravene har tilstrækkeligt syn og ikke lider af hindrende sygdomme der gør det umuligt at spille spillet.

4. Visuelle Krav
    a. Brikkerne skal tegnes med hashtags ✓
        i. Åben spillet, se på brikkerne
    b. Skal være konsol-baseret ✓
        i. Åben spillet, registrer at der er en konsol
    c. Baggrunden skal være sort, og brikkerne skal have forskellige farvers ✓
        i. Åben spillet, se om baggrunden er sort og brikkerne er forskellige farver
5. Informationsmæssige krav
    a. Den skal vise næste tetris stykke ✓
        i. Åben spillet, se om den viser det næste stykke
    b. Den skal vise score ✓
        i. Åben spillet, kig på scoren
    c. Vis hvor bevægende tetris stykke vil lande ✓
        i. Åben spillet, kig på spøgelsesstykket
6. Interaktionsmæssige krav
    a. Styres med piletasterne ✓
        i. Åben spillet, tryk på piletasterne og bevæg stykket
    b. Mellemrum skal rykke brikkerne ned ✓
        i. Åben spillet, tryk på mellemrum og se om stykket falder ned
    c. 'C' tasten skal gemme nuværende stykke ✓
        i. Åben spillet, tryk på C tasten og se om den bliver gemt

TEKNISKE DETALJER

Spillet i sig selv er som sagt programmeret i C++ (revision 17), og compilet til en Windows portable executable[18] ved hjælp af Microsoft Visual C++ [19] redskaberne (forkortet msvc), hvilket indebærer en proprietær compiler og en IDE kaldet Visual Studio. For at compile dette projekt kræves en C++ version 17 kompatibel compiler, men det er ikke garanteret, at det vil virke da Microsoft Visual C++ compileren ikke direkte følger C++ standarden, hvilket er en skam da det udelukker projekter fra at blive spredt til andre platforme med andre compilere. Jeg har testet projektet i Visual Studio Community 2017 version 15.6.4 men burde være kompatibel med enhver version af Visual Studio Community 2017. Den vedhæftede portable executable (tetris.exe) er statically-linked[20], så den kan køre på alle Windows maskiner så længde operativsystemets arkitektur er kompatibel med programmets arkitektur. Tetris spillet er compilet som et 64-bit program, da størstedelen af Windows maskiner nu kører 64-bit[21]. 32-bit programmer er selvfølgelig mere kompatible end 64-bit programmer grundet emulation, men emulation er en kæmpe flaskehals når det kommer til præstation, da hver gang man vil køre en WinApi funktion, skal der navigeres i gennem en masse forskellige libraries, hvilket i den lange ende trækker hastigheden af programmet ned drastisk.

[18] https://en.wikipedia.org/wiki/Portable_Executable
[19] https://en.wikipedia.org/wiki/Microsoft_Visual_C++
[20] En proces hvor i stedet for at libraries bliver loadet når programmet bliver kørt, er de refererede funktioner og datastrukturer implementeret ind i portable executable filen, dette gør filen meget større end normalt, men sikrer sig at computere der ikke har de henholdsvise libraries kan køre programmet.
[21] http://store.steampowered.com/hwsurvey/?platform=pc

OPTIMERINGER

Det er vigtigt for et spil at køre med høj præstation, og det er et af de største problemer indenfor spil-industrien. Mit spil er blevet testet på operativsystemet Windows 10, 64-bit både på en gammel bærbar og en rimelig ny stationær computer med hvad der ville blive klassificeret som 'high-end' specifikationer. Jeg kørte adskillige test runs for at få den optimale præstation, og jeg formod at få programmet til at køre op til otte gange så hurtigt ved at omskrive konsol-brugeroverfladen til at bruge en form for buffer, der hver frame tjekkede om et felt i konsollen enten ændrede farve eller karakter, så den ikke tegnede på felter der ikke havde ændret sig siden sidste frame. Mit benchmark[22] fungerede ved at måle tiden det tog at lave alle udregningerne og tegne alle felterne i alle frames, og så tage gennemsnittet af det. Første benchmark lå på 5700 µs (mikrosekunder), hvilket lyder som lidt, men det kunne jeg sagtens gøre hurtigere. Jeg compilede projektet med en masse specifikke compiler indstillinger der fokuserer mere på hurtigere maskinkode end færre maskinkode. Indstillingerne var således:

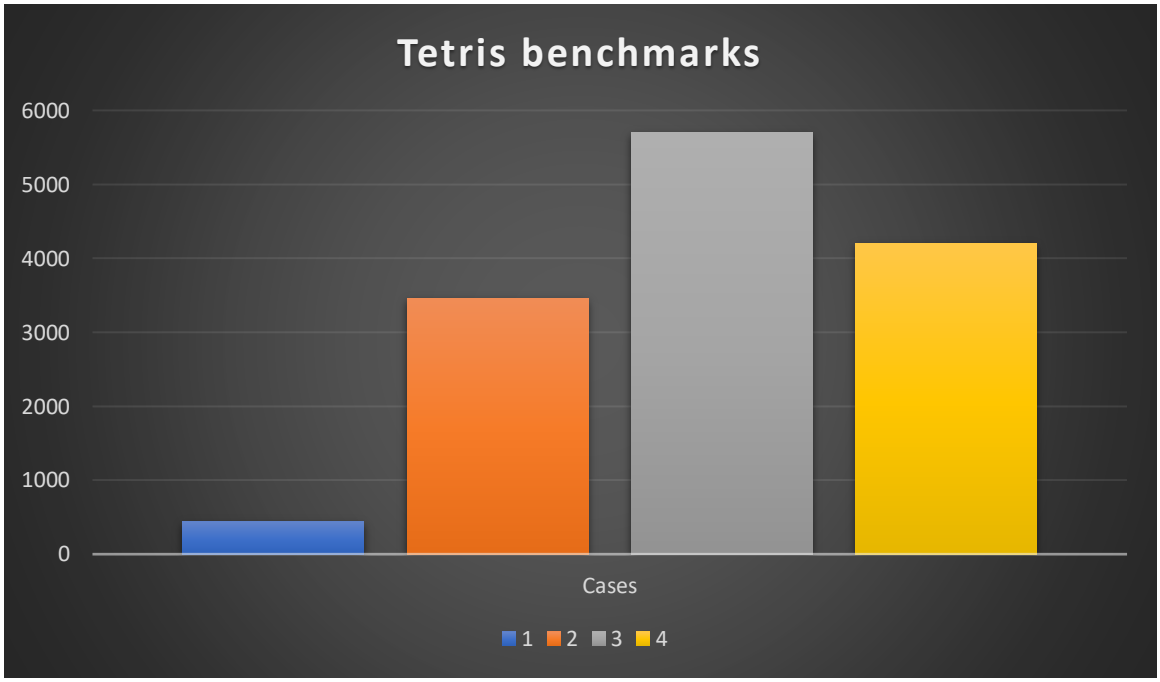
1. /O2 **'**Maximum Optimization (Favor Speed)**'** er en indstilling der fortæller compileren at den generalt skal foretrække præstation og hastighed over mængden af maskinkoden
2. /Ot **'**Enable Intrinsic Functions (Yes)**'** er en indstilling der fortæller compileren at den skal 'inline', det ved sige indskrive funktionen direkte ind i koden i stedet for at oprette en instruktion der kører. Dette resulterer i meget længere maskinkode, men også maskinkode med højere præstation da du sparer nogle instruktioner.
3. /Oi **'**Favor fast code**'** er en indstilling der fortæller compileren at når den optimere programmet (se /GL), skal den foretrække hastighed over alting, lidt ligesom /O2 (Maximum Optimization (Favor Speed))
4. /GL **'**Whole Program Optimization (Yes)**'** er en indstilling der fortæller compileren at den skal optimere hele programmet, hvilket både gør programmet hurtigere men i mange tilfælde også gør at den fylder mindre.

[22] https://da.wikipedia.org/wiki/Benchmark

Jeg kørte fire forskellige benchmarks for at teste mine optimeringer. Disse benchmarks blev målt ved hjælp af standard klassen '*std::chrono*'[23] og dens '*steady_clock*'[24] klasse. Jeg har brugt denne klasse da den tillader timing af maskinkode helt ned på nanosekund-niveauet, i modsætning til andre måder som tick-count[25] måling der er begrænset til millisekunder. Alt i alt er jeg meget tilfreds med mine optimeringer, konsol-brugeroverfladen var en flaskehals til at starte med, men jeg lavede den fuldstændig om så den brugte buffer-logik så den ikke spildte cpu cycles[26] på at skrive til felter som ikke burde skrives til.

Her ses en graf af de henholdsvise 'cases', hvor tiden er målt i µs (mikrosekunder).

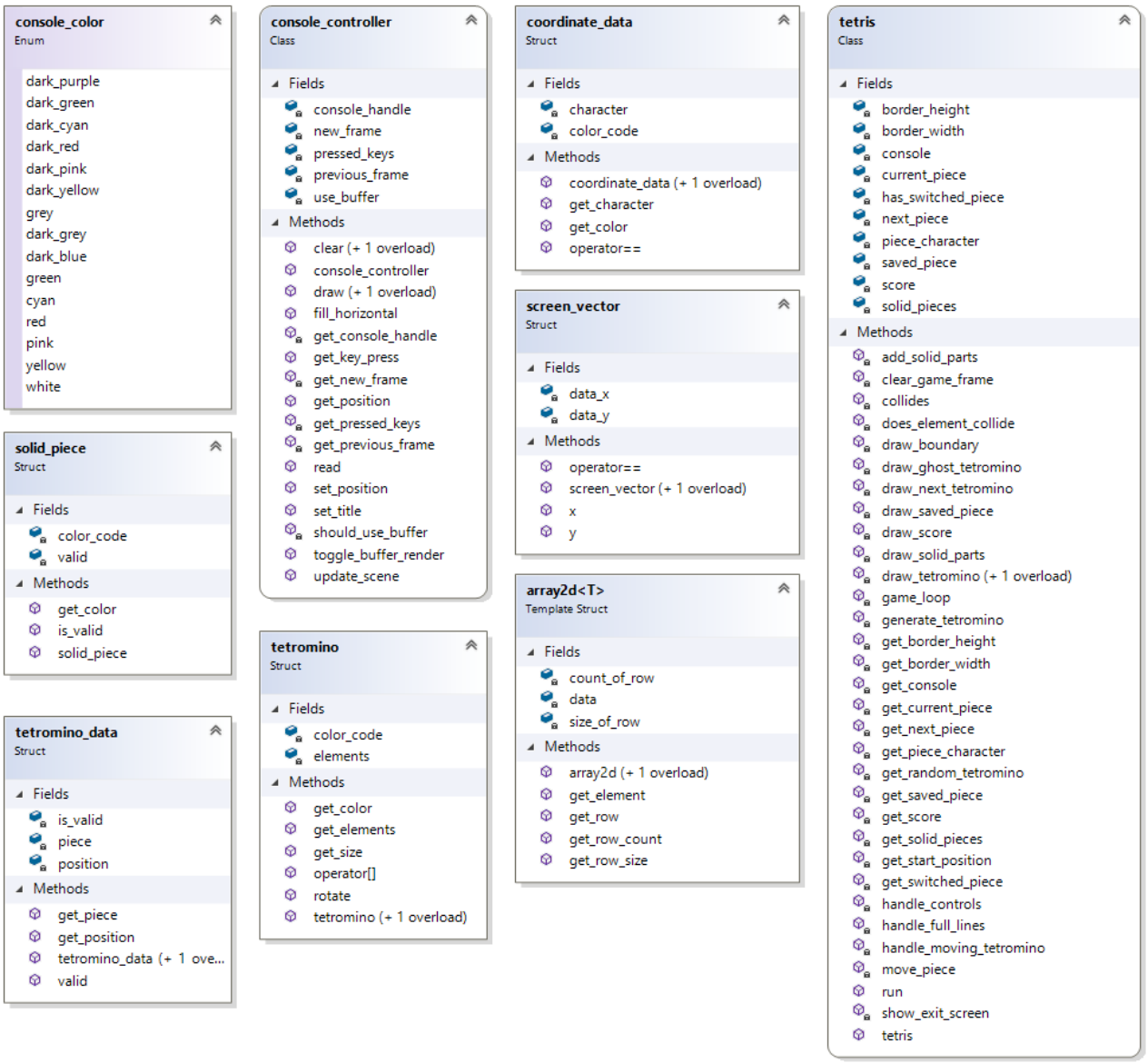
1. 440 µs (med buffer og compiler indstillinger)
2. 3460 µs (uden buffer, med compiler indstillinger)
3. 5700 µs (med buffer, uden compiler indstillinger)
4. 4200 µs (uden buffer, uden compiler indstillinger)



[23] http://en.cppreference.com/w/cpp/chrono
[24] http://en.cppreference.com/w/cpp/chrono/steady_clock
[25] https://en.wikipedia.org/wiki/System_time
[26] https://en.wikipedia.org/wiki/Instruction_cycle

FILER

Mit projekt består af 9 header filer og 7 source filer. Header filernes filtype er .hpp da det er standard header filtypen som bruges i c++ for at skælne mellem normale C headere (.h). Jeg har kun tænkt mig at gå over header filerne, da source filerne trods alt bare indeholder funktionsdefinitionerne fra de henholdsvise klasser. Her er et klasse-diagram der er genereret af Visual Studio, det gør det nemt at få et overblik over projektet, det viser klasserne og deres henholdsvise medlemmer og funktioner



Header filer:

1. 'array2d.hpp'

Array2d er min egen en implementation af et dynamisk to-dimensionelt array[27] ved hjælp af en '*std::vector*'[28], som er en form for liste af elementer (præcist ligesom et array, bare dynamisk i modsætning til standard arrays). Array2d er en templated klasse, hvilket vil sige at typen af elementerne ikke er fast defineret, hvilket tillader mig at lave two-dimensionelle arrays af alle former for klasser og objekter. Jeg bruger Array2d til at gemme og læse felter i konsollen, da en skærm kan deles op som et to-dimensionelt array af pixels, eller felter i mit tilfælde.

2. 'console_controller.hpp'

Console_controller er min egen, hjemmelavede brugeroverflade for windows konsollen. Den gør det nemt at skrive, læse og ellers modificere Windows' standardkonsol. Brugeroverfladen tillader at skrive tekst og tegn i forskellige farver (de understøttede, hard-codede farver som Windows nu har valgt at bruge, de har gemt i console_color). Brugeroverfladen er lavet fra bunden med præstation som første-prioritet, og brugervenlighed som anden-prioritet. Efter at eksperimentere med forskellige løsninger, valgte jeg til sidst at bruge en form for buffer logik, der igennem det to-dimensionelle array gemmer alle felterne, og deres henholdsvise karakter og farve. Hver frame skal funktionen '*update_scene*' køres, for at opdatere felter der er blevet ændret på. Den henholdsvise opdateringsfunktion går i gennem det to-dimensionelle array og sammenligner nuværende frame med det tidligere, og tegner alle ændringer, derefter gemmer den nuværende frame som tidliger frame. Brug af buffer logikken er selvfølgelig valgfrig, og kan styres af brugeren ved at køre funktionen '*toggle_buffer_render*' og dermed ændre indstillingen. Internt bruger brugeroverfladen funktionerne '*std::printf*'[29] og '*FillConsoleOutputCharacterW*'[30], der henholdsvis skriver en eller flere karaktere til konsolen, og fylder konsolen med en given karakter. Konsolen implementerer også en keyboard handler, der læser og udregner om en kiven tastaturknap er blevet trykket ned, og ikke blev trykket sidste gang funktionen blev kørt. Dette gør, at man kan implementere en input handler der tjekker om en knap er trykket, men ikke holdt nede. Input handleren er implementeret i tetris klassen.

[27] https://processing.org/tutorials/2darray/
[28] http://en.cppreference.com/w/cpp/container/vector
[29] http://en.cppreference.com/w/cpp/io/c/fprintf
[30] https://docs.microsoft.com/en-us/windows/console/fillconsoleoutputcharacter

3. 'coordinate_data.hpp'

Coordinate_data er en simpel struktur, der indeholder og gemmer karakter og farve for et givent felt, den bruges i konsol brugeroverfladen til at passere information i mellem de henholdsvise funktioner der står for at håndtere og tegne felter i konsolen. Den er meget kort og bruges enlig bare for at gøre koden lettere at læse.

4. 'rng.hpp'

Rng er et namespace, i modsætning til alle de andre filer, der implementerer en pseudo-random tal generator, det vil sige en funktion der giver tilstrækkeligt tilfældige tal. Mere specifikt bruger den '*std::mt19937*'[31], '*std::random_device*'[32] og '*std::uniform_int_distribution*'[33] til at implementere en tal-generator algoritme ved navn Mersenne Twister[34], hvilket er den mest udbredte tilfældighedsgenerator, der er blevet brugt til at generere tilfældige, kryptografisk sikre tal siden den blev opfundet i 1997. Jeg bruger denne implementation i stedet for '*std::rand*'[35] som man tit ser bliver brugt, men den er erklæret usikker[36] og burde ikke bruges i nogle programmer overhovedet.

5. 'screen_vector.hpp'

Screen_vector er endnu en simpel struktur, der indeholder x og y koordinater for et givent punkt på skærmen. Det er enlig bare en to-dimensionel vektor, men jeg har valgt at kalde den screen_vector for at gøre det nemmere for læseren at forstå dens egenskab.

6. 'solid_piece.hpp'

Solid_piece er også en simpel struktur, der enlig bare indeholder en boolean om den er gyldig og en farvekode. Denne bliver brugt i listen af solide stykker der er sat fast på skærmen når den bevægende brik bliver sat fast på grund af collision.

[31] http://en.cppreference.com/w/cpp/numeric/random
[32] http://en.cppreference.com/w/cpp/numeric/random/random_device
[33] http://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
[34] https://en.wikipedia.org/wiki/Mersenne_Twister
[35] http://en.cppreference.com/w/cpp/numeric/random/rand
[36] https://crypto.stackexchange.com/questions/52000/how-can-c-rand-be-exploited-if-a-secure-seed-is-used

7. **'tetris.hpp'**

Dette er hoved headeren der implementer selve spil-logikken. Der er en kæmpe rækkevide af funktioner i denne header fil, alt fra collision-detection til input håndtering. Input håndteringen er opsat som et '*std::map*'[37] af anonyme funktioner, hvilket er en lækker c++ egenskab der gør det super nemt at oprette callback tables som denne. Den indeholder også selve spil loopet og de interne spil strukturer. Denne klasses logik er beskrevet senere i flow-diagramet.

8. **'tetromino.hpp'**

Dette er en middel-stor struktur der definerer et tetris stykke. Det implementerer en liste af stykker der tegnes ved hjælp af relative (til nul) vektorere, og en rotationsfunktion der drejer tetris stykket 90 grader med uret.

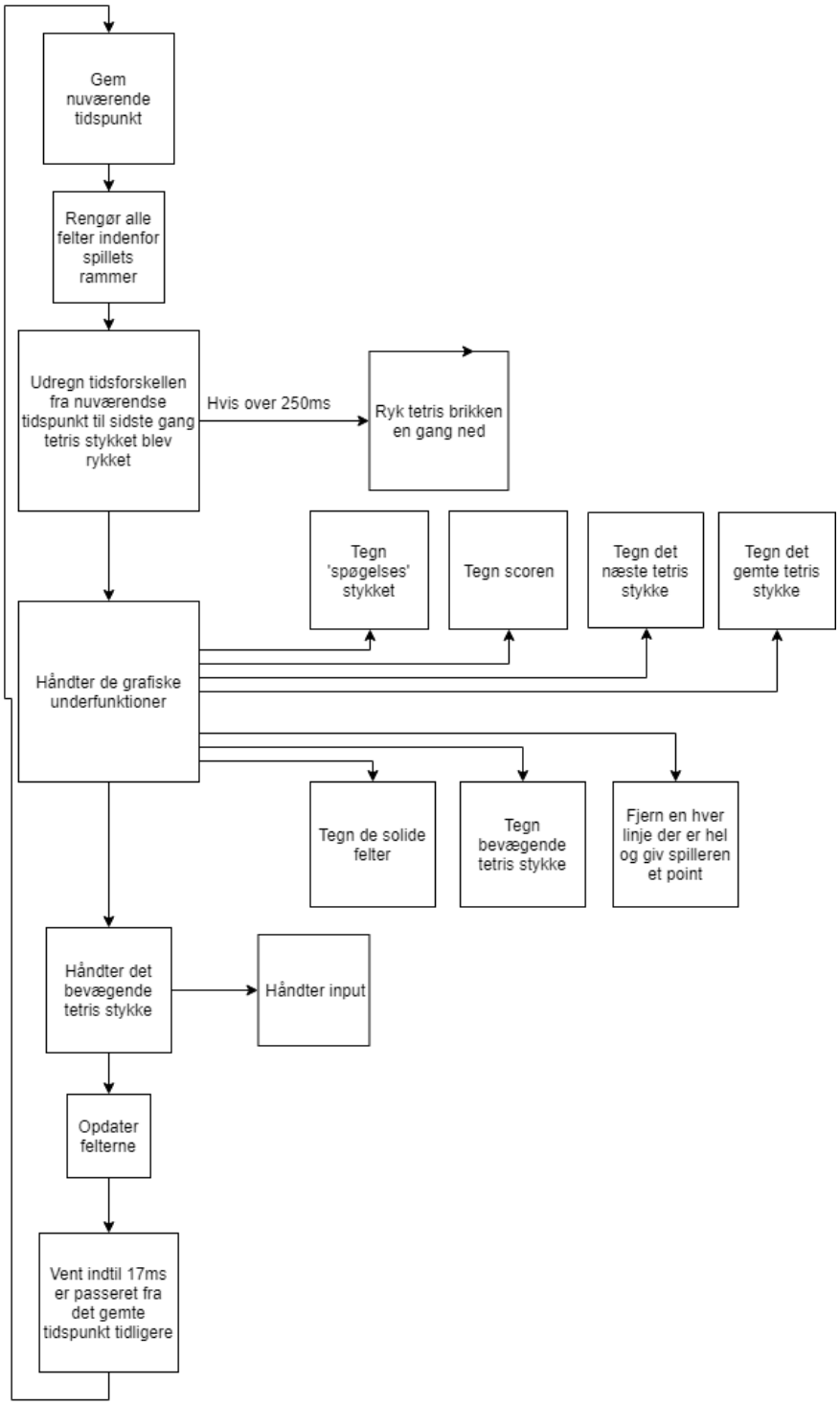
9. **'tetromino_data.hpp'**

Tetromino_data er en simpel struktur der indeholder tetris stykket, skærm position og information om tetris stykket er gyldigt.

[37] http://en.cppreference.com/w/cpp/container/map

FLOW

Jeg har valgt at visualisere programmets flow og logik, da det gør det nemmere at forstå hvad der helt præcist sker uden at læse flere hundrede linjer kode. Det er meget tidskrævende og kan virke forvirrende hvis jeg tegnede flowdiagrammer for alle funktioner i projektet, så jeg har valgt kun at lave et af selve render loopet, som bliver kørt efter at konsolen er blevet sat op.



FEEDBACK

Jeg har valgt at spørge adskillige personer om deres holdning for at få et indtryk af hvordan projektet havde udfoldet sig.

Simon Larseng Bang

> Rigtig god kopi, mangler dog at kunne holde piletasterne inde, men ellers meget godt :D

Yusuf Baysuz

> Super fedt spil, kan ikke helt se hvad det skulle mangle

Samuel Rasmussen

> Det her er rigtig cute :3 fedt spil

Anton 'Osten' Daal

> Fint spil, er dog ikke fan af konsol ting tingen men styretasterne er intuitive så man regner det hurtigt ud. Jeg ville dog ønske at det gemte min highscore