

1 Innhold

1	Innhold	1
2	Innledning	2
3	Utstyrliste	2
4	Problembeskrivelse.....	2
5	Løsningsbeskrivelse	3
5.1	Brett og Matrise	4
5.2	Orm	4
5.3	HighScore.....	5
5.4	Spill.....	6
5.4.1	Konstanter	6
5.4.2	Konstruktør	6
5.4.3	Start, menyExit, menyInfo og menyHighScore.....	6
5.4.4	Spill.....	7
6	Installasjon og instruksjonstutorial	9
7	To do, or not to do.....	9
8	Utleddning.....	10
8.1	Fremgangsmåte og tid:	10
8.2	Problemer	10
8.3	Konklusjon.....	10
9	Referanser & Kilder.....	10
10	Vedlegg	10

2 Innledning

Denne oppgaven går ut på å teste vår evne til å programmere et større system i C++. Systemet vi skal programmere er spillet Snake, som burde være kjent for de fleste.

3 Utstysrliste

- Microsoft Visual Studio .NET 2003
- Macromedia Captivate
- Microsoft Word 2003
- Microsoft Office Document Imaging
- Microsoft MSN Messenger

4 Problembeskrivelse

Denne oppgaven går altså ut på å programmere snake. Spillet består av en bane, der en styrer en slange ved hjelp av piltastene. På banen skal det være mat, og når slangen spiser/treffer denne maten, så skal slangen vokse, og spilleren skal motta en sum poeng. Banen skal aldri bli tom for mat, og den skal naturligvis ikke dukke opp utenfor banen eller oppå for eksempel slangen eller ting som allerede ligger på brettet. Spillet avsluttes hvis slangen treffer en vegg (f.eks kanten på banen) eller sin egen kropp.

Det jeg trenger å finne ut av og å lage er altså:

- Noe som holder styr på hva som er på banen av mat, vegger, etc
- Noe som til en hver tid vet hvor slangen befinner seg på brettet
- Noe som gjør at brukeren kan kontrollere slangen
- Noe som undersøker hva slangen treffer, og gjør handling deretter

Når det gjelder koden, så er målet å lage en løsning som ikke inneholder feil, eller som tar hånd om feil om de skulle oppstå. Den skal være oversiktlig, godt strukturert og ikke minst godt dokumentert. Vi bør dessuten legge til noen utvidelser av spillet i tillegg til minste kravet. Jeg har valgt å legge til følgende:

- Overraskelses mat (får tilfeldige ting til å dukke opp på brettet)
 - Tilfeldige vegger

- Udødelighetsmat, der brukeren for en kort periode ikke kan tape (det vil si at brukeren hvis brukeren treffer en kant vegg, spretter ormen bare tilbake, brukeren kan kjøre igjennom seg selv, og slangen kan "spise" de tilfeldige veggene som eventuelt har dukket opp)
- Mat som halverer lengden til ormen, men som også øker farten på spillet betraktelig for en kort periode (slik at det er positivt, men også litt risky å ta dem)
 - Spesial og overraskelses maten gir også mer poeng enn vanlig mat.
 - En gradvis fartsøkning mens man spiller, slik at det blir vanskeligere og vanskeligere. (Den øker for hvert 10. mat en spiser)
 - Poenggivning som står i forhold til hvor fort spillet går. Med andre ord vil man ikke få forskjellig poengsum for typer forskjellige epler. (Altså vil man få mer poeng for hver mat når farten øker)
 - Highscore liste som lagres til fil
 - (Sikkert et par ting til som jeg har glemt fordi de har blitt en så naturlig del av spillet etter å ha spilt og programmert på det en stund....)
 - Har forøvrig også valgt å kalle spillet for Ultra Worm 2.0 istedet for Snake (2.0 er fordi jeg startet med en versjon, men kastet alt og begynte på nytt).

Har også for morroskyld lagd en installasjon til spillet, slik at det kan installeres og avinstalleres. Med installasjonen vil det også bli lagt en snarvei i start-menyen, og en vil kunne ta en instruksjons tutorial på hvordan å konfigurere console vinduet (laget med Macromedia Captivate).

5 Løsningsbeskrivelse

Jeg skal nå beskrive min løsning. Og måten jeg har tenkt å gjøre det på, er å først si litt om klassene som brukes i spillet, og så beskrive spillet til slutt. Skal også si litt om hvorfor jeg har valgt å gjøre ting som jeg har gjort.

Sånn generelt sett har jeg valgt å la spillet bestå av `char` og `COORD`. `COORD` er en struct som defineres i `windows.h`. Den brukes av for eksempel

`SetConsoleCursorPosition()` metoden som naturligvis blir brukt mye i en console versjon av snake. Å benytte `COORD` structen i stedet for en egen punkt klasse eller slikt,

gjorde det dermed mye enklere å skrive ut i de posisjonene en skulle og alt slikt. Jeg har brukt forskjellige `char` til å beskrive de forskjellige elementene i spillet. På den måten blir det igjen enkelt å skrive de ut i consolet, samt kjapt og enkelt å sammenligne de med andre `chars`. Men nå over til hvordan jeg har valgt å lagre disse `char`'ene.

5.1 Brett og Matrise

Fra starten av visste jeg at det jeg var ute etter var en eller annen type av 2-dimensjonal array. Prøvde meg på å bruke en vanlig 2-dimensjonal `char` array, men fikk det ikke til uten å surre masse med structer/pekere eller slikt, og jeg ønsket meg en enkel og forståelig løsning. Så jeg googlet på nettet, og fant en tråd i et forum som omhandlet nettopp dette. Der var det et eksempel på en matrise (som jo er en 2-dimensjonal array), og det er den jeg har brukt til å lagre `char`'ene i brettet. Matrisen er en enkel template-klasse. Den bruker en `vector` med `r` `vector`er i seg som har plass til `k` elementer hver. På den måten får vi på en måte en matrise som er `r*k` stor. Klassen har også definert `[]` operatoren slik at en kan skrive for eksempel `liste[5][2]` og få det som ligger i punkt (5,2).

Brettet er da det som holder styr på denne matrisen. Det er også brettet som hos meg legger `char`'er på tilfeldige steder. Spillet gir bare rettet den `char` som skal plasseres, og så prøver brettet seg frem til det finner en ledig plass (som ledig plass har jeg valgt å bruke `space`, ' ', slik at et tomt sted tilsvarer det som generelt sett er sett på som en tom rute på skjermen. Derfor fylles også matrisen med `space` `char`'er når brettet initialiseres). Brettet har også en versjon av denne plasserings metoden, som kan sjekke opp mot et `orm` objekt, slik at den ikke legger noe i en rute som ormen befinner seg i. Det er vel stort sett det som er å si om brettet. Det har naturligvis noen flere funksjoner, som å fjerne et felt, sjekke et felt, gi/sette størrelsen sin og slikt, men det vil jeg påstå det er lettere å se på i selve kildekoden.

5.2 Orm

Siden console, og derfor mitt spill, bygger på `COORD` har jeg valgt å la ormen være en samlig `COORD`er. På den måten vil det være enkelt å for eksempel hente ut haletipp `COORD`en, og så skrive en `space` i den `COORD`en når halen skal følge etter ormen.

Jeg har valgt å bruke en `queue` til å lagre disse `COORD`ene. Dette fordi ormen kan sees på som en First-In-First-Out rekke med koordinater. Når man skal flytte på ormen vil man hele tiden legge på en bit (altså en `COORD`), og så fjerne den som har vært der lengst (med andre ord den som ble lagt på først). Noe som er litt forvirrende med dette er at fronten av køen vil være halen, og bakerst i køen vil være hodet. Men takket være innkapsling er det jo bare orm objektet som trenger å bry seg om dette, siden spillet utenfor uansett bare vil kalle `vokse()` og `krympe()` metodene.

5.3 HighScore

Filer og C++ er noe jeg aldri har hatt helt grepet på. Har prøvd å skjønne det lenge, men har aldri funnet noen gode eksempler på hvordan å gjøre det. Denne gangen fant jeg derimot et par eksempler på hvordan en kunne skrive en struct rett over i en binær-fil, for så å hente den tilbake igjen. Så måten jeg har valgt å lagre en highscore på er derfor ved hjelp av en `score-struct` som inneholder en `int` for poeng, og en `char[30]` for navnet. Lagringen foregår ved at den skriver 10 slike etter hverandre i en binærfil. Siden en `int` og en `char[30]` har faste størrelser så vil også structene har faste størrelser. Som en følge av de faste størrelsene kan den så lese de 10 `score-struct`ene inn igjen fra fila uten problemer. Begge operasjonene foregår med samme nærmest samme forløkke. Forskjellen er at den skriver og leser.

Ellers har jeg valgt å la `highscore` klassen ha en metode som oppretter en default `highscore` liste. `HighScore` klassen inneholder en `score-array` med 10 `score-struct`er. Hvis den ikke finner den fila som er angitt i klassen, vil den opprette en default liste istedet, og opprette en fil med den istedet.

Måten å legge til en score på er å opprette en struct med poengsummen og navnet, og så sende den som parameter til `leggTil()` metoden i `HighScore`. Denne vil starte nederst i listen, og så eventuelt flytte den oppover til den finner en som er høyere. Den bruker altså en slags bubblesort metode. Bubble-sort er vel den mest ueffektive sorterings metoden, men siden listen alltid vil være sortert fra før av, og det kun er en score som skal plasseres, så fungerer det fint. Vet forøvrig ikke om noen annen smart måte å gjøre dette på. Måten det fungerer på nå er at den starter å bunn, sammenligner med den over, og hvis den over er mindre, bytter de plass, og ser så på neste. Den nederste vil ganske enkelt falle ut av listen og forsvinne.

5.4 Spill

Dette er hovedklassen i mitt Ultra Snake 2.0 spill. Main funksjonen gjør altså ikke annet enn å opprette et Spill objekt, og så sparke det igang med `start()` metoden. Den inneholder både meny-funksjonene og selve spill-funksjonene. Mulig jeg burde splittet det opp i flere klasser, men siden dette er ét spill har jeg valgt å la det være i en klasse. Hadde spillet vært større ville det naturligvis vært en annen sak, men sånn som det er nå vil det etter min mening føre med seg mer problemer enn positive ting å splitte det opp. For det første ville det bli mer komplisert å skulle bruke verdier fra spillet i menyene, og en ville måtte drive å sende masse ting frem og tilbake. For det andre har vi dette med gjenbruk av kode, og klasser er jo ofte skrevet for at en skal kunne bruke de om igjen. Som for eksempel matrise klassen `min`. Spillet derimot kan uansett ikke brukes til noe annet enn akkurat det den er skrevet for. Menyen kan heller ikke brukes til noe annet enn akkurat dette spillet, så å skulle dytte det ut i en annen klasse er ikke helt nødvendig. Men nok om det, over til hvordan det hele fungerer. Måten jeg skal gjøre det på, tror jeg er å beskrive hovedpunktene.

5.4.1 Konstanter

Bare for å ha nevnt det, så har jeg latt alle de forskjellige typene ting som dukker opp på brettet, være definert som konstante `char`'er. Da kan jeg bruke navet på konstanten når jeg beskriver tingen i info-skjermen og skriver dem ut, og på den måten vil det være enkelt å skifte dersom jeg finner ut at jeg vil at for eksempel maten skal se anderledes ut.

5.4.2 Konstruktør

Denne starter ikke selve spillet, men fikser litt forskjellig rundt om kring. Den setter for eksempel størrelsen på vinduet, størrelsen på outputbufferen, fargen, etc. Den deaktiverer også det som gjør at den automatisk skifter linje etter input, skriver ut det som blir skrevet inn, og automatisk går på ny linje når den når slutten av linjen. Altså deaktiverer `ENABLE_LINE_INPUT`, `ENABLE_ECHO_INPUT` og `ENABLE_WRAP_AT_EOL_OUTPUT`.

5.4.3 Start, menyExit, menyInfo og menyHighScore

Start er en løkke. Den går uendelig. Det den gjør er å kalle `meny()` metoden, den skriver ut en meny, og returnerer en `char`. Start tar så denne `char`, og kaller en annen

metode, ettersom hva som ble valgt. Ble det valgt noe den ikke kjenner igjen, tar den bare menyen en gang til. At `start()` metoden kaller de andre metodene resulterer også i at den alltid til syvende og sist ender tilbake i menyen, hvilket er det som er målet med en meny. Av ting som kan velges er: `menyInfo()` som vil skrive ut hvordan man spiller og slikt, og returnere tilbake til menyen. `MenyHighScore()` vil skrive ut highscoren, og returnere til menyen. Siden start kjører uendelig, vil `menyExit` kalle `exit()` metoden som avslutter programmet i stedet for bare å returnere. Siste valget er `spill()`. Dette er selve snake spillet, og den vil jeg nå beskrive hvordan fungerer:

5.4.4 Spill

Hvordan spillet henger sammen er det egentlig lettere å se ved å følge kildekoden. Den skal både være godt kommentert, og relativt oversiktlig og grei. Men jeg bør vel si litt om hvordan jeg har valgt å løse de spesielle problemene som er nevnt i problembeskrivelsen. Første tingen som bør kommenteres er at jeg har valgt å la spillet ha ansvaret for all utskrift på skjermen (hvilket er grunnen til at jeg har definert char for de forskjellige tingene i spill klassen og ikke i brett eller orm. Med andre ord er altså brett kun en lagring av char, og orm kun en lagring av `COORD`.

5.4.4.1 Problem 1: Styring av ormen

Etter litt erfaring med animering i Flash, brukte jeg en liten teknikk overført derfra. Du har en variabel som holder styr på hvor du er (dvs. `x` og `y`, men her bare en `COORD`) og to variable kalt `xrate` og `yrate`. `xrate` og `yrate` blir for hver gjennomgang lagt til `x` og `y`. På den måten er det eneste en trenger å gjøre, å forandre på `xrate` og `yrate`. Og det er det jeg gjør. Starter med å sjekke for tastetrykk, og hvis for eksempel vestre piltast er trykket ned, vil `xrate` settes til `-1` og `yrate` til `0`. Så når da plasserings `COORD`en oppdateres, vil dens `y` ikke forandres, mens den vil flyttes en til venstre i `x` retning.

5.4.4.2 Problem 2: Holde styr på hvor ormen er

Videre vil så denne nye tilstands koordinaten legges til i ormobjektet, pluss at det vil tegnes en `char` for ormen i det punktet. For å få halen til å følge etter henter jeg ut (og altså fjerner) en `COORD` fra orm objektet, og skriver ut en space på det stedet. Ormen har også en metode som tar inn en `COORD` og går gjennom seg selv og sjekker om den har

denne `COORDEN`. Denne brukes for eksempel av brettet når de skal finne en ledig plass til hva den nå enn skal plassere ut.

5.4.4.3 Problem 3: Finne ut hva ormen treffer

Dette gjøres ganske enkelt ved å sende `COORDEN` som hodet til ormen er i til brettet, og så få returnert `char`'en som er der. Hvis det er en blank space, har den ikke truffet noe. Hvis det da for eksempel er en char som tilsvarer mat `char`'en, så vil den gjøre deretter.

5.4.4.4 Problem 4: Få ormen til å vokse

Som løsning på dette har jeg å bruke en vokse variabel. Hvis denne er mer enn null, så lar den være få halen til å følge etter, og minker istedet bare denne variabelen med 1. Så det som skjer når brukeren treffer et eple, er at denne vokse variabelen økes med 2, noe som resulterer i at ormens hale vil henge igjen 2 gjennomganger av spill-løkken. Når vokse variabelen er nede på 0 igjen vil halen fortsette å følge etter og ormen vil være 2 lengre.

5.4.4.5 Problem 5: Spesial maten

Teller teknikken som jeg har brukt til maten har jeg også brukt til de andre spesial tingene i spillet. For eksempel når den sjekker om du treffer deg selv eller veggen, sjekker den også om udødelighetsteller variabelen er 0. For hvis den er det så dør du, hvis den derimot er mer enn 0, vil andre ting skje. Hvis den treffer en kant vegg vil for eksempel `xrate` og `yrate` reverseres, slik at man spretter ut fra veggen igjen.

5.4.4.6 Problem 6: Øking av farten og poenggivning

Jeg har latt poengene for maten være i forhold til farten i spillet. Farten bestemmes av en pause i starten av løkken. I begynnelsen er den på 100ms. Og den minkes for hver n'te mat den spiser, ned til en "maksfart" på 30ms (har selv testa, og det er vanskelig, men mulig. 20ms derimot, som jeg hadde først, var litt `_for_vanskelig`). Poengene en får for hver mat bestemmes så ved at jeg tar defaultverdien, 100ms, og trekker fra den farten den har for øyeblikket. Legger også til 1, slik at en alltid får minst 1 poeng. Er foreksempel farten i spillet for øyeblikket på 80ms, vil spilleren få $20+1$ poeng for maten som spises. (Hvis jeg ikke legger til 1 vil brukeren ikke få noen poeng for de 10 første som spises, og det ville jo vært litt kjedelig...). Jeg har latt spesial tingene også gi poeng, men da disse er litt spesielle, har jeg latt de gi kvadratet av den differansen + 1. Altså

hvis farten er 85, vil poengsummen bli $5*5+1=26$. På den måten blir de litt ekstra attraktive i tillegg til det de ellers gjør.

5.4.4.7 Problem 7: Avslutning av spillet (når du dør altså)

Dette var ikke et stort problem, men tenkte jeg kunne nevne det for ordens skyld. Spille kjører i en `while(alive) { }` løkke. Hvis en av funksjonene som sjekker om du har truffet en vegg eller deg selv eller slikt, finner ut at du har gjort nettopp det, så setter de den til false og hopper opp til sjekken igjen med `continue`. Løkken vil da være ferdig og kjøre `skrivHighScore()` funksjonen til slutt. `skrivHighScore` funksjonen sjekker om scoren er høy nok, og hvis den er det henter den inn brukerens navn, og sender videre til highscoren. Til slutt vil de returnere til menyen der det hele startet

6 Installasjon og instruksjonstutorial

Da jeg fant ut at Visual Studio har en samling med Setup and Deployment Projects var jeg bare nødt til å teste det ut. Litt unødvendig for et program som består av kun en programfil, men synes det var gøy å prøve. På den måten vil brukeren kunne installere og avinstallere det som vi er vant til å kunne gjøre. Vil også få en snarvei i startmenyen sin. For morro skyld har jeg også prøvd meg i programmet Macromedia Captivate. Dette er et program for å lage instruksjons presentasjoner og slike ting. Den har den muligheten at den kan ta opp det du gjør, og så lage presentasjon ut i fra det. Har ikke hatt noe lurt å prøve det ut på før, men kom på at jeg her kunne lage en for hvordan å forandre instillingene til console vinduet. Det vil si hvordan å skifte fargene, og fontstørrelsen.

7 To do, or not to do...

Jeg vurderte en stund å lage et objekt for ting som ligger på brettet. Kunne da hatt et felles interface med ting som `getChar` og slikt, og så laget et object for mat og de andre tingene som da fulgte det interfacet. Men det lot jeg være da dette bare ville ført til problemer og styr som strengt tatt ikke er nødvendig. Kunne også for eksempel ha lagt menyen inn i en egen klasse, eller latt den ligge i main metoden, men føler det er litt unødvendig å dele opp hvis hensikten bare er å dele opp ting for å dele opp ting.

8 Utledning

8.1 Fremgangsmåte og tid:

Startet med å få til å styre en strek som startet på midten av skjermen, og ble lenger å lenger. Så fikk jeg halen til å følge etter, og pga måten jeg har løst det på, så var det etter det stort sett bare morro.

8.2 Problemer

Har ikke hatt noen ekstremt store problemer i denne oppgaven, selv om det selvfølgelig alltid er små ting en stanger litt i men kommer seg som regel igjennom det uten større vanskeligheter. En ting som er verdt å nevne derimot er kanskje at jeg lenge slet med at ormen min ikke kunne ta krappe svinger (altså at hvis man prøvde å snu 180* ville ormen gå først to ned før den svingte, istedet for at den la seg helt inntill seg selv som den nå gjør). Og dessuten hvis man trykket veldig fort(eller holdt en tast nede), endte ormen opp med en plagsom forsinkelse (altså at den utførte ting senere). Dette gjorde det naturligvis vanskeligere å spille, så lot det en stund bare være en del av spillet, selv om jeg hele tiden lurte. Men til slutt fant jeg ut av det, og grunnen er den at piltastene returnerer 2 verdier og ikke 1 som de vanlige bokstav-tastene gjør. Først en verdi som sier at det er en funksjonstast(hvis jeg har forstått riktig) og så selve verdien på tasten. Løsningen var altså å filtrere ut den unyttige verdien, og nå fungerer det ypperlig.

8.3 Konklusjon

Utrolig artig og lærerik oppgave! Synes det er litt vanskelig å få til mye objektorientering i et lite spill som snake (ender opp med at det blir mer innviklet, spredt og uoversiktlig enn det trenger å være), men det er jo nettopp noe av det vi skal trene oss på c",)

9 Referanser & Kilder

- C++ kompendiet og forelesningsnotater
- <http://www.codeguru.com/forum/showthread.php?s=&threadid=231046>

10 Vedlegg

- Kildekode, kompiert program og installasjon(med instruksjons film)