UiT
NORGES
ARKTISKE
UNIVERSITET

INF-1100 Oblig 4.

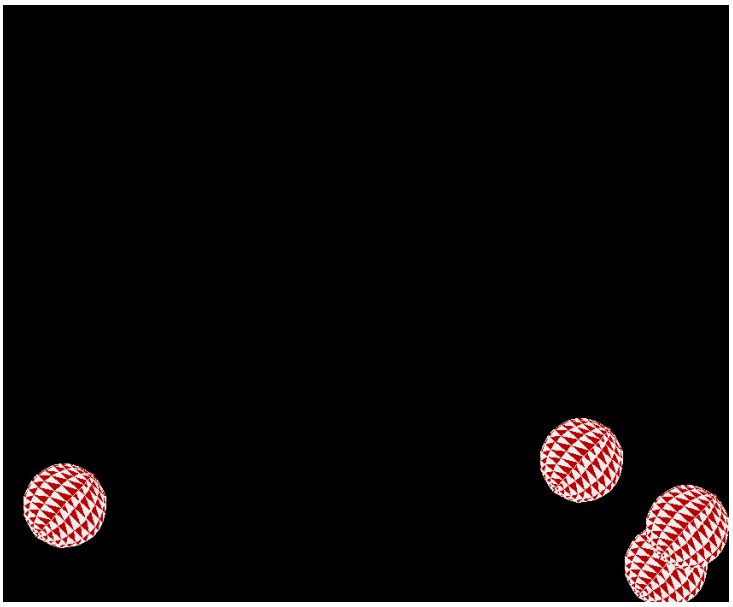
Fysikk Bachelor
18. November 2016
Martin Soria Røvang

1. Introduksjon

I denne oppgaven skal det sprettes baller med et realistisk gravitasjonsfelt

Her lærer man å bruke andre koder og programmer, hvordan man bruker linked list[4] og minne allokering.

Når alt er gjort riktig så skal jeg få noe slik.



Figur 1.

1.1. Krav

Forstå lenka liste[4] med iteratorer[1] og bruke dynamisk minne (malloc), her skal det lages liste med noder og kunne fri de løs fra minnet igjen. Her er det også ekstremt mye bruk av pekere.

2. Teknisk bakgrunn

Her brukes SDL (Simple Direct media Layer) dette er et program som lar meg få grafikk ut av C koden.

Her brukes også lenka liste[4] som har noder[2] som inneholder datamengder. I dette tilfelle informasjon om hvordan et objekt skal virke ved å ha angitte hastigheter, størrelse på ball og en array med informasjon til hvordan selve ball modellen skal se ut.

Dynamisk minne for å kunne lage plass til all informasjonen og kunne frigi de igjen for å spare plass. Dette er veldig viktig når det er mye som skal skje for å ikke kreve enormt mye fra pcen til unødvendig lagring.

3. Oppbygging av programmet

Her vil det ramses opp alle funksjonene som måtte bli laget, dermed vil det diskuteres slavisk hva hver vil gjøre.

List.c.

- List_create

Denne funksjonen lager en tom liste som er klar til å bli fylt med med noder.[2]

- List_destroy

Denne funksjonen skal slette hele lista.

- List_addfirst

Denne funksjonen skal legge til ny node i lista og den skal komme først i i rekka hver gang denne blir brukt.

- List_addlast

Denne funksjonen legger noden bakerst i lista.

- List_remove

Denne funksjonen sletter items fra lista og dermed minker list_size.

- List_size

Returnerer antall baller (objekter) i lista.

- List_iterator_t[1]

Lager en iterator liste.

- List_destroyiterator

Tømmer iterator lista fra minne.

- List_next

Peker seg igjennom noden og returnerer ballen den var på.

- List_resetiterator

Resetter iteratoren slik at den peker på første noden i listen.

Object.c

- CreateObject

Denne funksjonen skal lage ballene, her må informasjonen til ballen ligge som hvilken model den bruker, hastigheten osv.

- DestroyObject

Funksjon for å slette minne til modellen og minne til informasjonen til ballen.

- DrawObject

Funksjon som skal tegne objektet, her plasseres model arrayen inn og en drawtriangle funksjon for å tegne ballen.

Main.c

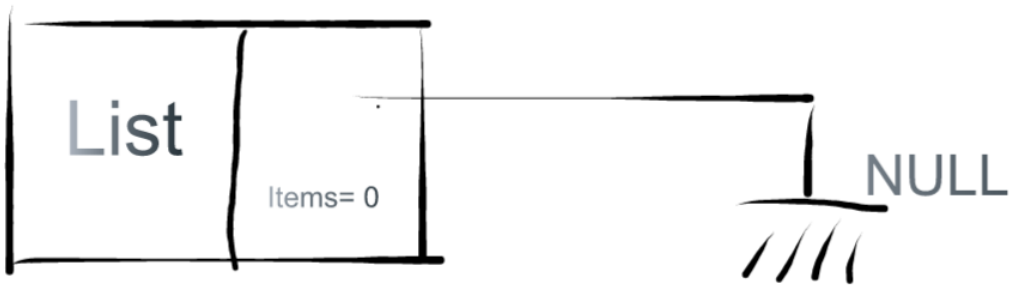
- Bouncingballs

Denne funksjonen skal få baller til å sprette i et gravitasjonsfelt ved hjelp av listene. Her blir den største koden å ligge.

4. Diskusjon

Det beste er å starte med listene først vil man lage list_create funksjonen.

Den skal lage plass i minne og lage den må «rengjøres» ved å NULLE ut adressa og gi informasjonen i lista verdien 0 slik at den ikke tar tilfeldig informasjon som ligger der minne blir plassert og dermed ikke skaper krøll senere. Denne funksjonen skal så returnere verdiene tilbake slik at det kan bli brukt av en annen funksjon.



Figur 1. *Tom liste* [4]

list_destroy skal tømme lista fra minnet fordi da bruker ikke pcen opp minnet som ikke er i bruk til noe, denne skal da brukes når programmet skal avsluttes og legger til en printf som sier at rensing av minne fungerte som det skal. Hvis denne funksjonen blir kjørt imens programmet er i gang vil hele programmet krasje fordi du fjerner noe programmet krever for å fungere.

List_addfirst skal lage den første noden i rekke den skal først sette av minne for noden også skal den plasseres først ved å legge den forrige noden bak seg og koble seg til head. Denne noden skal også bli gitt en item som da er informasjonen til objektet. Her er det også lagt inn en sjekk som ser om det er plass i minne, hvis dette ikke er sant så avsluttes programmet. Tilslutt så blir det lagt på en i lista får å vise at det er en ball i liste, denne blir plusset opp for hver ball.

List_addlast legger en node bakerst i rekka, denne skal da også lage minne til den nye noden, men denne funksjonen må lete seg igjennom lista med node helt til den finner den som er bakerst i lista.

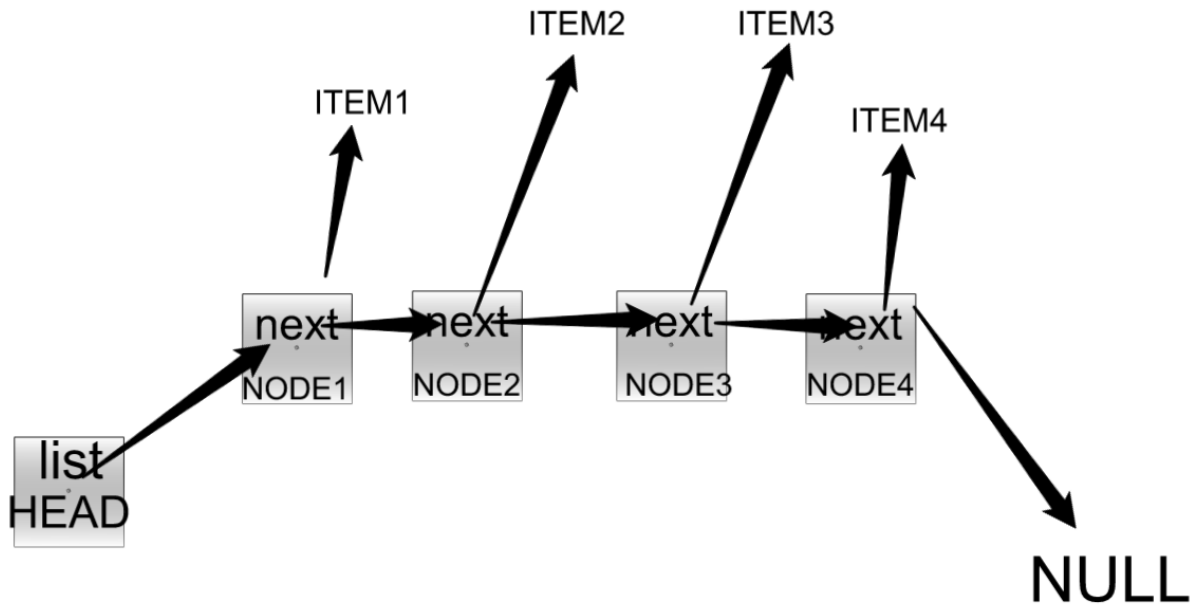
Siden alle noder som er siste må være kobla til en NULL (tom adresse) her kobler man alle ender for å ha styr på hvor noe slutter, slik at f.eks når du blar igjennom med iteratoren så vil den komme til en slutt slik at den vet når den kan stoppe. Og får å lett slette all minne du bruker.

Da du blar igjennom får å finne den siste noden så skal den slutte når den finner en node med NULL

Da skal den ta forrige peker og slenge den på seg og så koble seg selv på NULL. Deretter slette NULL fra den som nå ligger foran seg.

Men siden alle nye noder er koblet til null fra før via lista så trenger man ikke bruke denne funksjonen.

Figur 2 viser hvordan det ser ut i liste strukturen



Figur 2.

Alle nodene har hver sin unike item fordi struct_listnode lager nye hver gang en ny node blir laget.

List_remove funksjonen skal fjerne en item fra lista altså all informasjonen til ballen, alt av hastighet osv.

Funksjonen lager først en listnode som leter etter item i lista og en if statement i funksjonen sjekker om den har funnet det, ved funn skal den slette itemet også skal funksjonen slette noden. Dette gjøres fordi hvis noden slettes først så vil man miste «tilkoblingen» til itemet og det vil ligge fritt i minnet.

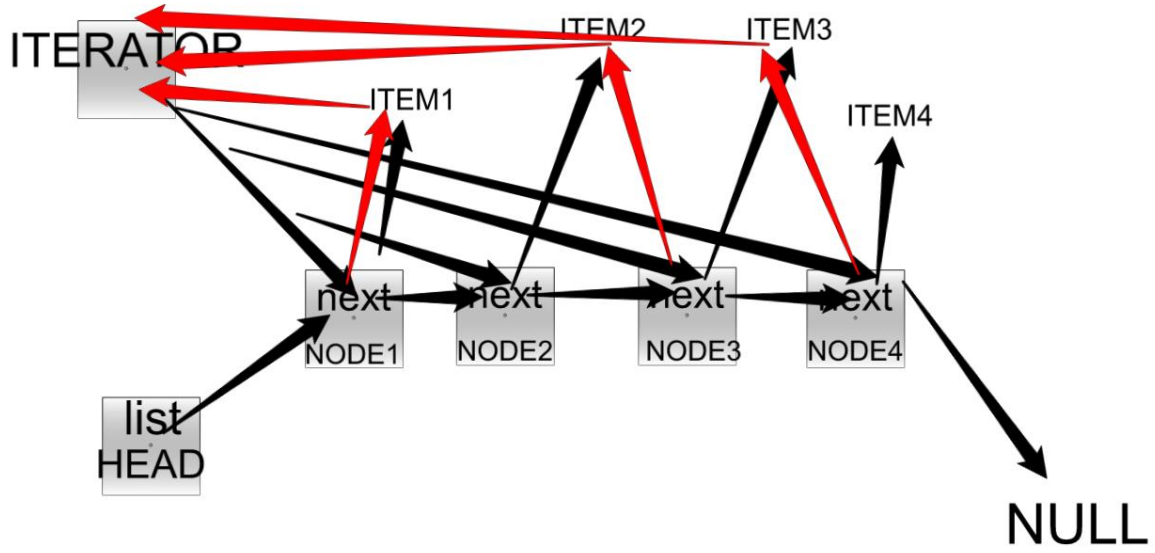
List_size er en ganske enkel funksjon, denne skal bare returnere hvor mange baller som er i lista for at andre funksjoner kan holde styr på hvor mange baller som skal komme inn om gangen.

List_createiterator skal gi minneplass til en iterator som starter på første node i lista

Denne skal da starte på head og returnere verdiene.

List_destroyiterator funksjonen skal frigjøre minne som brukes av iteratoren, denne skal brukes ved avslutting av programmet sammen med list_destroy.

List_next denne funksjonen skal iterere igjennom alle nodene i lista som vist i graf 3



Graf 3.

I iteratoren skal det være en item peker som er en void[3] for å kunne ta inn hvilken som helst verdi (char, int osv) denne skal returneres hver gang den går til neste node.

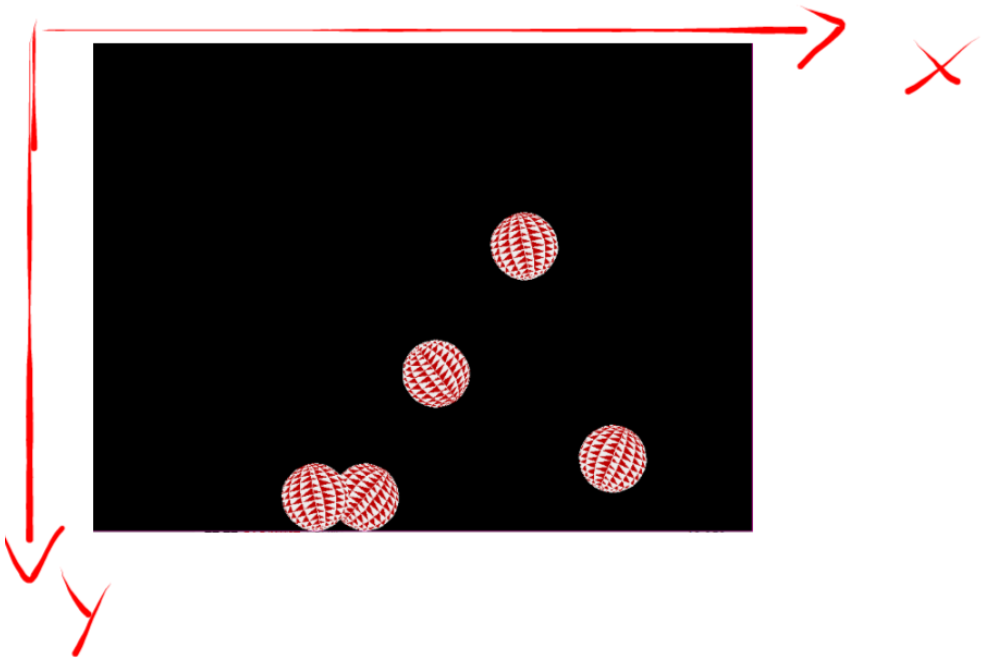
List_reset iterator skal sende iteratoren tilbake til første node i lista ved å sette iteratoren til å være head dette må gjøres fordi iteratoren stanser alltid på den siste noden (når det leser av NULL) så for at det skal være en kontinuerlig bevegelse så må denne til.

Createobject som er i object.c er ballen i programmet. Her allokeres minne og blir gitt en model, hastighet og levetid. Her har det også blitt lagt inn en random funksjon for å gi ballen tilfeldig hastighet i x retning hver gang en ball blir laget. Dette gjør at alle ballene ikke ender med å bare ligge oppå hverandre og lage mer variasjon i hvor ballene lander.

I denne funksjonen ligger også startposisjonen til ballen, og skalaen, tilslutt skal funksjonen returnere verdiene til at det kan bli henta i en annen funksjon.

DrawObject funksjonen henter all informasjon fra model array til ballen slik at at verdiene kan manipuleres som rotasjon og størrelser. Etter henting av all informasjon om modelstrukturen så tegnes objektet med drawtriangle funksjonen, dette er så satt i en loop for å få ut alle trekantene til modellen.

Bouncingballs er den største funksjonen og skal styre alt av bevegelser og hvordan ballen skal oppføre seg. Her vil ballen havne inni et rom der ballen starter med en positiv hastighet som da virker nedover fordi koordinat systemet til pcen er i tredjekvadrant slik graf 4 viser.



Graf 4.

Så når ballen treffer bakken så skal hastigheten ganges med et negativ tall for å få en motsatt rettet hastighet som virker da oppover. Denne ble satt 0.9 for at ballen skal miste 10% av sin hastighet for hver gang den treffer bakken slik at det virker realistisk, dette gjelder også veggen.

Hoveddelen av bouncingballs funksjonen er i en nested whileloop dette må til for at det skal være en kontinuerlig forandring av ballenes posisjon.

Utafor loopene har jeg noen konstanter for å lage en realistisk bevegelse av ballene slik at de triller bortover og mister energi.

Her lager jeg også lista med list_create som da lager den tomme lista deretter er det en iterator liste funksjon. Her er det også plassert en liten whileløkka som lager 5 baller som da vil være de første ballene som kommer når man åpner programmet disse er kalt inn via en funksjon som sender inn vilkårlig objekt informasjon som er kalt «addtoscreen» dette er fordi det er mange måter ballene kan bli lagd i dette programmet. Slik som start ballene, baller ved å trykke på «a» og tekanner ved å trykke på «w». Det er også en som sender inn baller hver gang det er færre enn 5 baller.

Den ytterste loopen skal rense skjermen og restarte iteratoren med list_resetiterator funksjonen som ble laget i liste header fila fordi i den innerste whileloopen så blir iteratoren «brukt opp» og havner helt bakert i lista med noder så dette er veldig viktig for at noe skal kunne bevege seg.

I den ytterste er det også switch funksjoner ved bruk av taster. Her er det mulig å trykke «a» for å lage flere baller (maks grense på 15 baller) og «w» for å lage små tekanner som oppfører seg som baller. Det er også mulig å trykke «Escape» for å avslutte programmet med exit(1). Under escape funksjonen så renser jeg også minne helt ved å bruke list_destroy og destroyiterator funksjonene for å slette alt av minne som har blitt brukt. Det er veldig viktig at disse ikke blir brukt under bruk av programmet for da vil de slette minne som er i bruk og så vil programmet prøve å skrive til disse plassene og dermed vil alt det kræsje. Under escape funksjonen er det også lagt til en leaderboard funksjon som skriver ut hvor mange ganger ballene du hadde traff taket og lagrer dette i en tekstfil slik at man kan få se tidligere poeng og lagre nye.

I den innerste whileløkka skal være avhengig av at ballen er lik den neste iteratoren for at den skal holde seg i løkka, det første i løkka er en Drawobject funksjon. Denne tegner ballene som har blitt lagt inn i lista.

Deretter kommer Gravitasjon delen av løkka. Den plusser på litt hver gang løkka går slik at den blir «trukket nedover» (Positivt er nedover på pc koordinat system) dette må altså plusse på objektets y hastighet (objekt->speedy) slik at det påvirker hver sin unike ball.

Her er det også lagt inn tak og vegger slik at ballen spretter (gir motsatt rettet verdi med litt tap) hver gang de er borti dem. Når hastigheten blir 0 så skal en «timer» verdi bli gitt til objekt strukturen slik at man kan finne differansen mellom SDL_GetTicks klokka som er en klokke i SDL programmet som teller opp i millisekunder med engang programmet starter opp. Når denne differansen er funnet (hvis man skal ha 5 sekunder så blir det 5000) så skal list_remove og destroyobject funksjonene kjøre der list_remove først går inn i nodene og sletter verdiene til objektet også tar destroyobjekt og sletter hele modellen sånn at du ser at ballen blir borte. Deretter kommer addtoscreen funksjonen igjen for å fylle opp med nye baller.

5. Konklusjon

I denne oppgaven måtte jeg være veldig kreativ. Først måtte jeg studere all koden for å få en forståelse av hvordan koden var bygget og hva jeg kunne endre på for å gjøre oppgaven. Jeg måtte lese mye på nettet for å klare å løse linka liste. Oppgaven er veldig lærerik og den motiverer for å gjøre mer med den.

6. Kilder

http://www.cs.yale.edu/homes/aspnes/pinewiki/C(2f)Iterators.html , 13.11.2016

http://stackoverflow.com/questions/29433188/next-pointer-element-in-linked-list-structure , 14.11.2016

http://stackoverflow.com/questions/692564/concept-of-void-pointer-in-c-programming , 14.11.2016

https://www.cs.swarthmore.edu/~newhall/unixhelp/C_linkedlists.pdf, 14.11.2016