Fulin Halvorsen
Jørgen Karlsen
Alex Ahnon

Dokumentasjon til C++ eksamen

Prosjektoppgaven

Oppgaven går ut på at vi skal programmere en Software løsning i C++. Oppgaven består i planlegging, programmering og bygging av arkadespillet «Space Invaders». Alle prosjekt filer skal leveres i en ZIP fil. Oppgaven skal leveres innen klokken 12:00 lørdag 2018.05.02.

Gruppen består av:

Alex Ahnon – 2. år spill programmering
Jørgen Karlsen – 2. år spill programmering
Fulin Halvorsen – 2. år intelligente systemer

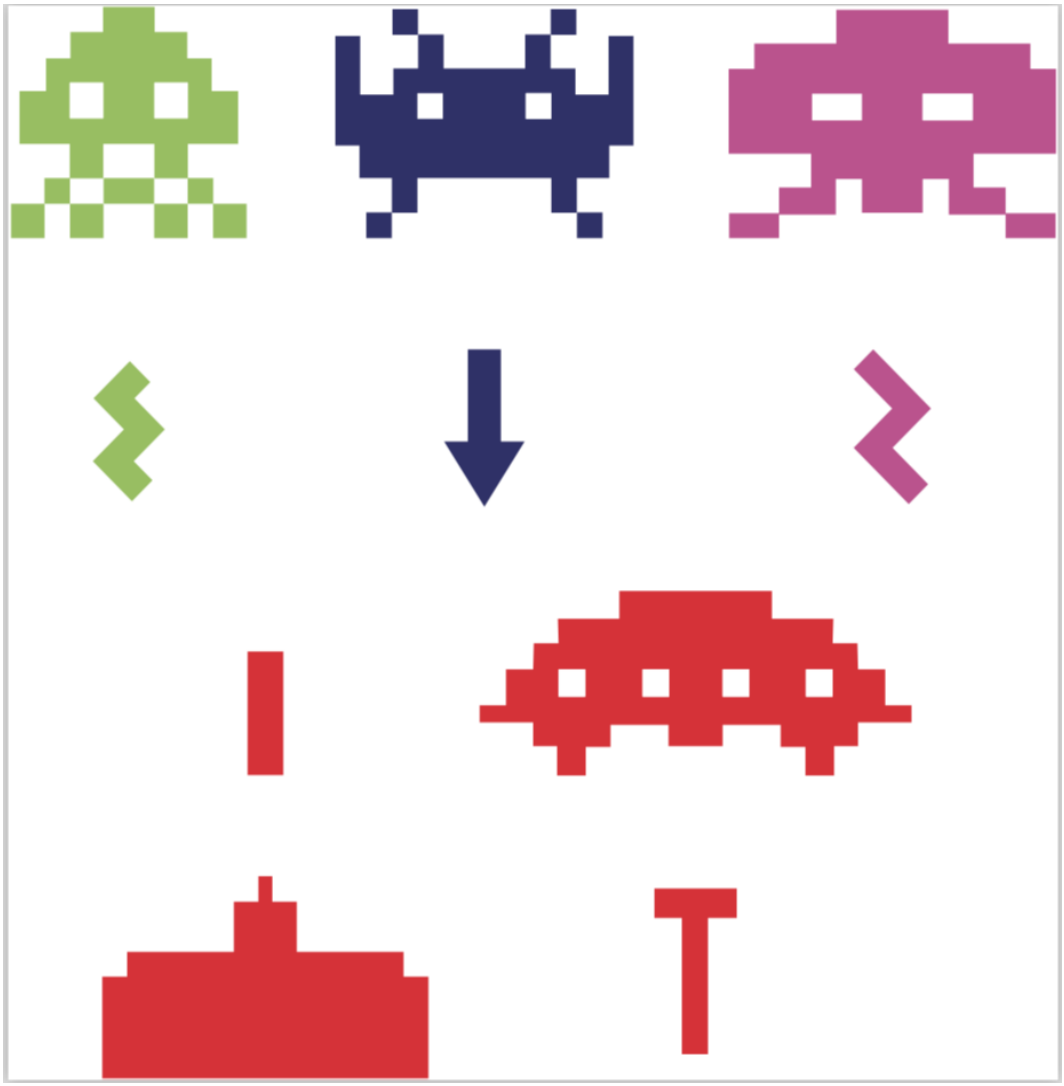
Tidsskjema:

2. mai 12:00 – 16:00	• Diskutere oppgaven • Fremgangsmåte • Opprettet Github Repository • Startet på dokumentasjon • Diverse ideer til løsningen • Jobbe med koden • Grafikk arbeid
3.mai 10:00 – 16:00	• Sette ikoner • Fortsette på dokumentasjon • Jobbe med koden
4. mai 10:00 – 16:00	• Jobbe med koden • Debugge • Testing av spillet • Dokumentasjon
5. mai 10:00 – 12:00	• Gjennomføre kvalitetssjekk • Dobbeltsjekke alle filer • Pakke inn i ZIP • Levere

Grafikk

Vi bestemte oss for å bruke det offisielle grafikken til spillet, vi lastet de ned og beskjærte alt i photoshop og konverterte filene fra PNG til BMP (Kilde 1). Vi prøvde å bruke PNG fremfor BMP for at bildene skal være helt transparent (diskutert videre i siste side), men for å gjøre det så ble det mer komplisert. Vi fokuserte heller på kravene først og prøvde å finne ut av det dersom vi hadde tid på slutten.

Den røde figuren med rødt prosjektil ble brukt for spiller-objektet, og den grønne figuren med grønt prosjektil ble brukt for alle enemies. Vi hadde ønsket å legge til variasjon på fiendene, men fikk uheldigvis ikke nok tid på slutten til å legge det til.



Gruppens samarbeid

Gruppen samarbeider bra, vi kjenner allerede hverandre fra før av fra Smidig prosjekt og via andre anledninger. Vi ble enig om å møte kl. 10 alle dager fremover bortsett fra første dagen.

Første dagen ble vi enig om å fordele oppgaver, vi sitter sammen og jobber for å holde god kommunikasjon og utnytter parprogrammering. Gruppen ble veldig fort enig om hvordan vi skulle fordele de ulike oppgavene, og det var viktig at dokumentasjonen ble startet på med en gang. På slutten av hver dag så blir gruppa enig om hva alle skal gjøre hjemme individuelt, så vi har jobbet både sammen på skolen og individuelt hjemme.

Under kodingen har vi møtt på mange utfordringer som vi ikke var helt forberedt på, alt fra store ting til små bagateller, men vi klarte å overkomme disse sammen.

Beskrivelser av klasser og viktige funksjoner

main

main blir brukt til å lage en instanse av GameManager og for å kjøre selve spillet.

WindowManager

WindowManager-klassen håndterer alt som har med å opprette vinduet å gjøre, og holder styr på variablene nødvendige for det. Det initialisere SDL gjennom sikker feilsjekking og returnerer riktige errors dersom det skjer feil. Denne klassen er en singleton siden vi krever kun en instanse. SetupWindow() går gjennom all initialiseringen, DeleteWindow() kjører alle nødvendige SDL destroy-funksjoner og UpdateWindow() oppdaterer vinduet.

InputManager

InputManager-klassen håndterer sjekking av keyboard state for å lage bool funksjoner som sjekker om en tast på tastaturet har blitt trukket, blir holdt nede, eller ble nettopp sluppet. Meste av koden kommer fra tidligere øvingsoppgave hvor vi fikk beskjed om å lage klassen. Vi bestemte oss for å la være å lage funksjoner for mus, og fokuserte kun på tastatur knapper. Denne klassen er en singleton siden vi krever kun en instanse.

GameManager

GameManager-klassen er hvor hoveddelen av spillet ligger og hvor alle klassene blir koblet sammen for å skape spillet. Denne klassen er en singleton siden vi krever kun en instanse.
Den har tre viktige funksjoner, Initialize(), Update() og Run(), med tilleggsfunksjoner som skal hjelpe med logikken til spillet. Selve GameManager-klassen er ganske stor, og det er mulig at det går ann å fordele den bedre i andre klasser.

I Initialize() funksjonen så initialiserer vi alle objektene våre og draw-er alle bildene til skjermen. Vi initialiserer også variabler og posisjoner av objektene, og alt annet som har med å gjøre før man kjører selve spillet.

Update() er funksjonen som ligger inne i game-loopen. Alle nødvendige funksjoner for å endre på objekter i runtime og sjekke for kollisjoner blir lagt inn her.

Run() starter spillet og kjører game-loopen, som inneholder Update() funksjonen og tar av seg viktige inputs som bevegelse til spilleren og skyting gjennom polling for events. Den utnytter InputManager for å kunne gjøre dette. Game-loopen har også en FPS regulator for å sikre at spillet kjører på en stabil 60 fps.

Sprite

Sprite-klassen er en viktig del av programmet og blir arvet av mange av de mindre klassene som Player og Enemy. Sprite håndterer alt som har med å tegne, renderer og forandre på selve figuren som vises på skjermen. Vi mente at dette var den beste måten å spare energi på, fordi flere av klassene skulle bli tegnet i vinduet og blitt håndtert gjennom SDL_Rect variabler for kollisjonssjekking.

Draw() tegner figuren på skjermen, RenderUpdate() oppdaterer objektet og Destroy() er litt eksperimentell, men skal fjerne teksturen til objektet og forandre lokasjon utenfor skjermen.

Text

Text-klassen er veldig lignede med Sprite og kunne sikkert ha vært kombinert, men vi bestemte oss for å dele den til sin egen klasse. Den gjør det samme som Sprite, men bruker heller SDL_ttf library pakken for å kunne tegne tekst objekter på skjermen. Update() funksjonen blir brukt til å oppdatere tekst i runtime.

Player

Player-klassen styrer variablene og objektet til spilleren. Den har en health variabel og en Update() funksjon som holder styr på cooldown-en for skytingen til spilleren. Har en MoveRight() og MoveLeft() funksjon som lar spilleren bevege seg på x-aksen. Arver fra Sprite-klassen.

Enemy

Lignede med Player-klassen, men for fiendene isteden. Update() funksjonen holder styr på cooldown for skytingen, men har også små AI som gir enemies ett fast mønster som de beveger seg gjennom på skjermen (Går fram og tilbake på x-aksen 3 ganger før den går ned et hakk). Har muligheten til å endre på timing og variablene for å endre vanskelighetsgrad. Arver fra Sprite-klassen.

Shield

Shield-klassen skaper objekter for skjoldene som skal beskytte spilleren fra innkommende skudd. Man setter maxHealth og initialXPos som blir brukt til å bestemme liv og posisjon. Har to funksjoner, TakeDamage() og Reset(). TakeDamage() gjør objektet 10 pixeller mindre i lengde, og fikser på posisjonen. Reset() resetter skjolded til orginale verdier. Arver fra Sprite-klassen.

Projectile

Projectile-klassen tar seg av alle prosjektilene som blir renderet i vinduet, både for spilleren og for fiendene. Den har to konstruktører, en for Player og en for Enemy, som bestemmer hvilke bilde den skal bruke, og om den skal skyte nedover eller oppover ved å sette CharacterType.

Planlegging, valg og hvordan vi taklet forskjellige krav

I begynnelsen av oppgaven så diskuterte vi som en gruppe på hvordan software løsningen skulle se ut. Vi ble fort enige om at vi ville fokusere på minimumskravene og lage et godt grunnlag for spillet for å gjøre programmeringen mye enklere for oss selv senere når vi skulle skape logikken.

Det første vi planlegde var hvordan vi skulle sette opp SDL vinduet og håndtere objektene som skulle bli rendret på skjermen. Dette endte opp med at vi kom på idéen av å ha forskjellige manager-klasser som skulle håndtere de forskjellige aspektene ved spillet. WindowManager håndterer det som har med vinduet å gjøre, GameManager håndterer update og game-loopen, og InputManager håndterer keyboard state.

Vi måtte også hå en måte å håndtere små-objektene som skulle bli rendret i vinduet. Vi lagde dermed en Sprite-klasse som håndterte variablene og funksjonene som var relevante for å skape et objekt på skjermen. Dette var for å redusere stor redudans av kode gjennom de mindre klassene som ble brukt for eksempler som spillereren, prosjektiler eller fiender.

I første omgang så ønsket vi å prøve å få til PNG-loading for bildene i spillet, sånn at de hadde alpha-channel og var transparent med bakgrunnen. Dette endte med å bli et stort surr av library dependencies og dll.feil, og uheldigvis endte ikke opp i den endelige løsningen for prosjektet (Vi prøvde oss fram med SDL_image pakken).

Når vi fikk alle klassene satt opp, så var det mye tankegang og diskusjoner på hvordan vi skulle håndtere logikken i game-loopen. Det endte med at mye ble lagt inn i Update() funksjoner til GameManager-klassen, og det krevde en del trial-and-error med vektorer, iterasjonsloops og testing av kollisjoner. I etterhånd, når vi ser tilbake etter at vi ble ferdige med minimumskravene, så skjønte vi at mye av koden kunne bli optimisert. Om vi hadde mer tid, så er det mye vi skulle ønsket vi kunne ha optimisert bedre.

Når vi skulle legge til et av tilleggskravene som innebærte at spilleren hadde skjold foran seg, så var vi usikre på hvordan vi skulle løse det. Vi så på orginal videoen og merket at deler av skjoldet forsvant når det ble skudd. Vi brukte litt tid på å se om vi kunne gjøre det samme, men endte ikke opp med å finne en lignede løsning. Isteden så krymper skjoldene i lengde når de blir skutt, og blir mindre og mindre til de forsvinner helt (tar 8 skudd totalt).

Den største hindringen vi omkom på var hvordan vi skulle få fiendene til å skyte prosjektiler mot spilleren, og hvordan vi skulle gjøre det mest mulig rettferdig uten at alle enemy-objektene (som da ligger i en vektor-array) skulle gjøre den samme aksjonen i hver update.

Valget vårt var å legge til en cooldown på hvor fort de kunne skyte, og brukte heller rand() for å gjøre det sånn at skuddene var tilfeldige for hvert individuel enemy. Vi satt sjansen til å være lav nok til at skuddene omkom ganske ofte, men ikke ofte nok at spilleren ikke kunne ungå dem.

Til slutt, siden vi hadde litt ekstra tid, så la vil til en tilleggsfunksjon hvor spilleren har 3 liv. Vi syntes dette var mer rettferdig og kunne øke vanskelighetsgraden på enemy waves (vi kalte det for wave isteden for bane/level). som ble spawnet etter den første. Liv og skjold blir reset etter hver wave.

Avslutning

Vi greide å fullføre alle kravene som ble gitt i eksamen, og var fornøyde med vår innsats og løsning ved avslutning.

Kilder

(1). http://www.classicgaming.cc/classics/space-invaders/graphics (Grafikk)