Yume Nikki Inspireret videospil

Process Rapport

Projekt af: Jack Conradsen

Lærer: John Ellehammer

Tec Ballerup

03-03-2023

1 Indholdsfortegnelse

2 Indledning

Mit projekt er stærkt inspireret af et allerede eksisterende videospil, *Yume Nikki*[1], Det er et indie spil udviklet i RPG-maker[2] engang i 2004, af en elusiv Japansk spil-udvikler, som går under aliaset *Kikiyama,* der vides ikke andet om udvikleren, og det er også det eneste spil de har udviklet.

Spillet fik en kult-følge online, og der eksistere nu over 1000 forskellige fan-spil lavet i samme stil. Nogle af disse fan-spil er der også taget inspiration af. En der er værd at nævne hedder
Yume 2kki[3]. Da det er den største og en af de ældste fan-spil lavet efter *Yume nikki,* det er samme premise, men i modsætning til en enkelt udvikler, er *Yume 2kki* igennem mange år blevet udvidet af over 100 forskellige udviklere, og bliver stadig videreudviklet i dag.

2.1 Hvad er Yume Nikki?

Man spiller som en pige, inde i et lille soveværelse, med få interaktive elementer, hvis man prøver at benytte døren til værelset, ryster karakteren bare på hovedet. Det eneste andede man kan gøre i værelset, er at spille et småt NES lignende spil på et lille tv, gå ud på balkonen til værelset, sidde ved et skrivebord, (som også gemmer spillet), eller gå i seng.

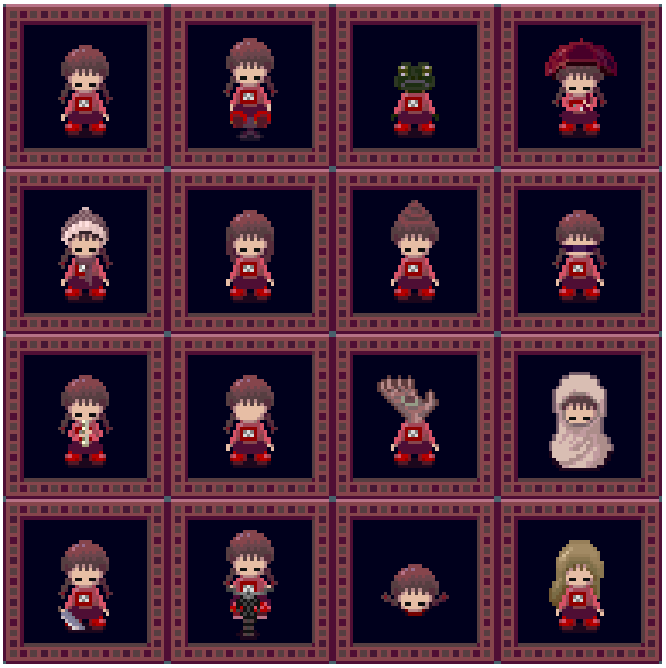
Yume Nikki - Pigens værelse

Hvis man går i seng bliver man ført til en drømmeverden, og det er her spillet rigtigt går i gang.

[1] Yume Nikki https://en.wikipedia.org/wiki/Yume_Nikki
[2] RPG-Maker https://en.wikipedia.org/wiki/RPG_Maker
[3] Yume 2kki https://yume2kki.fandom.com/wiki/Yume_2kki_Wiki

Der er en stor kollektion af forskellige abstrakte drømmeverdener[4], man kan udforske. Mange verdener har indgange til flere dybere verdener, og nogle looper tilbage til forrige verdener. Der er også unikke scriptede events der kan sættes i gang i nogle verdener, i nogle tilfælde helt tilfældigt, og andre gange efter meget specifikke kriterier.
I nogle verdener kan man også opsamle "effekter" som bliver en del af spillerens inventar, der så kan equippes og ændrer på karakterens udseende, verdens omgivelser, eller helt ændre gameplayet.

Yume Nikki - Forskellige effekter



Det er ikke et konkret mål i spillet, men hvis man får opsamlet alle 24 effekter i spillet, kan man opnå en slutning. Det er et ekstremt abstrakt spil, hvis indhold samt mystiske spil-udvikler har ført med til at folk stadig debattere hvad dets betydning er selv 20 år efter. Siden det handler om drømme verdener, er det et abstrakt syn ind i en karakters personlighed og historie, og alle der spiller det kan danne deres egen unikke ide om hvad det egentlig handler om.

2.2 Hvorfor lave endnu et fan-spil?

Yume Nikki trods sin abstrakte natur er en af mine favorit spil, da det er en af de ting som der i nogle tilfælde kan bruges længere tid på at diskutere og spekulere over end faktisk at spille.

Og dets premise med at udforske drømmeverdener, giver uendelige muligheder for at udvikle spændende scenarier og gameplay. Derudover er der ingen grænser for hvor

[4] Eksempler på drømme verdener i Yume Nikki se Billag/yume_nikki_eksempler

langt sådan et spil burde være. I løbet af denne opgave, vil jeg opnå mindst v.0.1 men kan efterfølgende blive ved med at bygge det videre.

2.3 Hvordan spilles spillet

Det skal understreges at det er en meget ukonventionel genre, så nogle vil måske betgene det mere som et interaktivt kunst stykke mere end et videospil.

2.3.1 Controls

Kontrols til spillet er meget enkle.

- Piletaster - bevæger spilleren og menu punkter
- Z / Enter - Ineregere med objekter og acceptere menu punkter
- X / Backspace - Åbner pause menuen, eller går tilbage i menuer
- 1 - Aktivere nogle effekters specielle egenskab
    - Pingvin effekten kan ligge sig på maven
    - TV effekten kan tændes og slukkes
- 9 - Spilleren dasker sig selv i hovedet og vågner

2.3.2 Hvad kan man gøre?

Herunder beskrives hvordan spillets forløb vil gå.

- Spillet startes. Der ses en start menu gui.

- Man kan starte et nyt save, eller loade et allerede eksisterende save.

- Man styrer en dreng inde på et soveværelse.

- Hvis man interagere med døren til værelset vil han ryste på hovedet

- Hvis man interagere med Computeren på skrivebordet, kan man gemme spillet.

- Hvis man interagere med sengen i værelset, går drengen i seng. Og efter kort tid bliver man teleporteret til en drømme version af værelset.

- Hvis man nu interagere med døren til værelset (mens man drømmer) kommer man til en verden fyldt med forskellige døre, der alle føre hen til andre unikke drømme verdener.

- I nogle af disse verdener kan man opsamle effekter, der så bliver tilføjet til ens effekt inventar.

- Hvis man går ind i inventaren og anvender en effekt, vil de ændre spillerens udseene, og gameplay.

- Der er f.eks en “Penguin” effekt. Der forvandler spilleren til en pingvin, og man kan så glide på maven og bevæge sig meget hurtigere.

- En anden effect der kan opsamles er “Stick” effekten, der gør spilleren ekstremt tynd, så man kan snævre sig igennem meget smalle åbninger.

- Man kan på ethvert tidspunkt mens man drømmer, trykke på en knap der for spilleren til at daske sig selv i hovedet, og så vågne op i soveværelset igen. Her kan man så igen gemme spillet ved computeren.

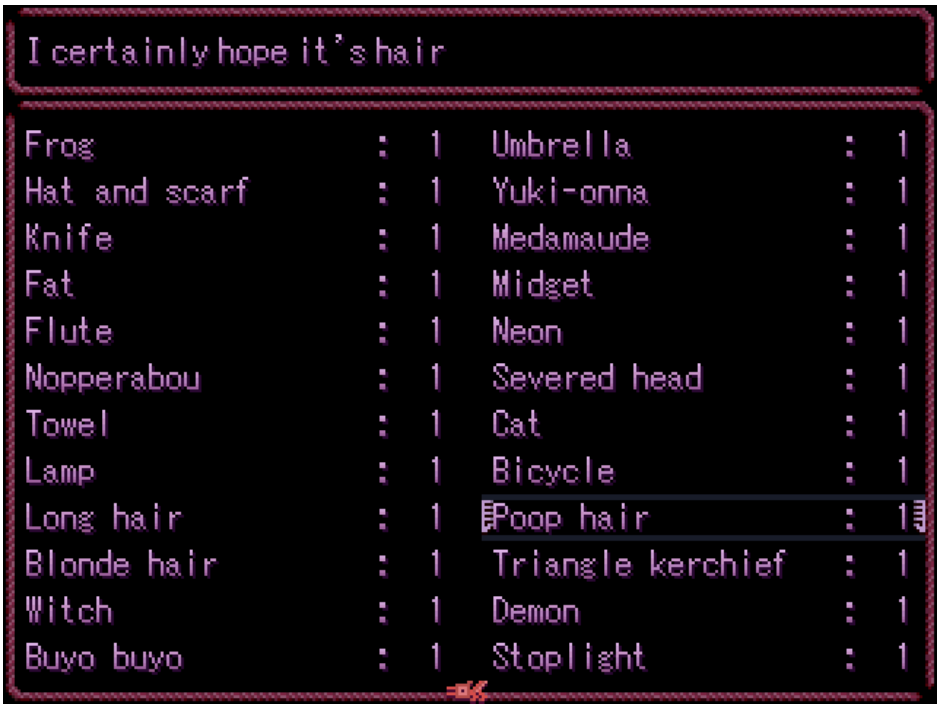
Spillet handler om at udforske disse mange drømmeverdener, opsamle effekter, og så benytte dem til at finde nye verdener

3 Problemformuleringsspørgsmål

Mine spørgsmål går på hvordan Yume Nikki’s elementer kunne blive rekreaert og eller forbedret.

1. Hvordan kan man bedst muligt gøre spilleren i stand til at anvende forskellige effekter, og genbruge kode? Mange effekter vil have nogle af de samme egenskaber

Yume Nikki - Effekt inventar



2. *Yume Nikki* blev originalt udgivet på et chat forum, eksklusivt på Japansk, det krævede at folk roede ned i koden for at modde engelsk sprog ind i det, før det blev kendt i vesten. Hvordan kan man gøre det nemt at have flere sprog, og tilføje flere sprog i fremtiden?

3. I *Yume Nikki* er mange af de forskellige baner meget store, og nogle gange uendelige, hvis man går ud til venstre, kommer man ind til højre. Hvordan kan man kode det og samtidig sørge for at det ser visuelt “seamless” ud?

4 Projektplanlægning

4.1 Estimeret tidsplan

Se Billag/Estimeret_tidsplan.pdf

4.2 Arbejdsfordeling

Under Eksamensforløbet har jeg arbejdet alene. Det har haft fordele og ulemper, men det var det bedste valg for at jeg kunne lave dette projekt, da der ikke er andre i min omkreds som har kendskab til Gamemaker, eller samme passion for at lave videospil. Så det gav mig mulighed for at fordybe mig i et emne jeg var interesseret i.

4.3 Backup / Github

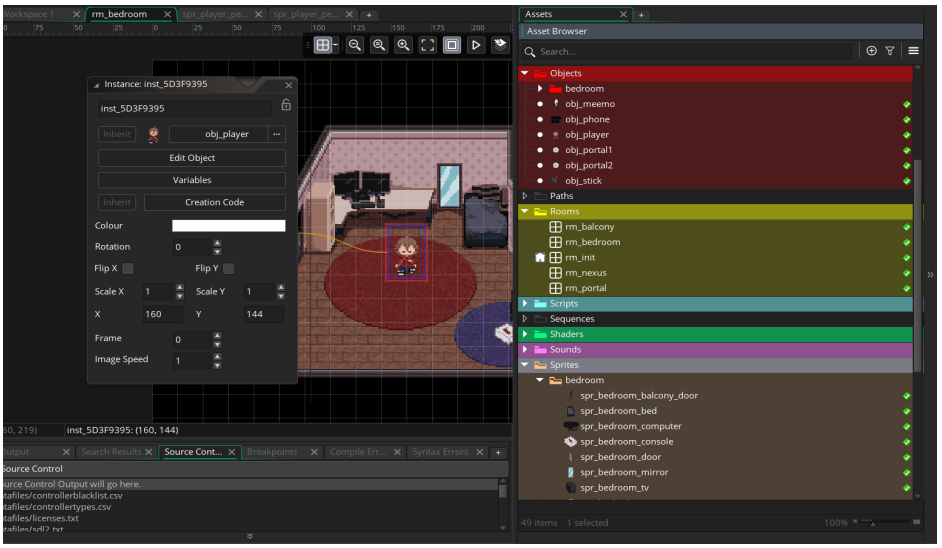
Jeg har i projektet benyttet Github til at bevare mine projekt filer. Siden jeg har arbejdet alene, har der ikke været nogen særlig måde at styre reviews og lign. Så det har mest fungeret som en online backup. Med få commit beskeder undervejs. Det gør det så også nemmere at kunne arbejde på en anden computer hvis det blev nødvendigt.

5 Hvorfor Gamemaker?

Gamemaker tager sig af mange af de ellers komplicerede og meget tunge krav, under spil udviklingen.

Skulle jeg F.eks. Udvikle spillet fra bunden i c++, ville jeg være nødsaget til at bruge det meste af udviklingstiden på at udvikle en game-engine til at rendere grafikken på skærmen, hvordan programmet modtager input asynkront, optimere vram, ram.
Alt det og mere tager Gamemaker sig af med det samme. Så det meste af udviklingstiden kan blive brugt på gameplay kode.

derudover er det også det sprog/program jeg har mest erfaring med, så der er ikke tidsspilde på at lære nye værktøjer at kende under udviklingen.



GamemakerStudio2 IDE

6 Hvordan fungerer Gamemaker?

En kort introduktion i hvordan Gamemaker fungere vil hjælpe til at forstå hvordan jeg har arbejdet på mit spil.

6.1 Assets

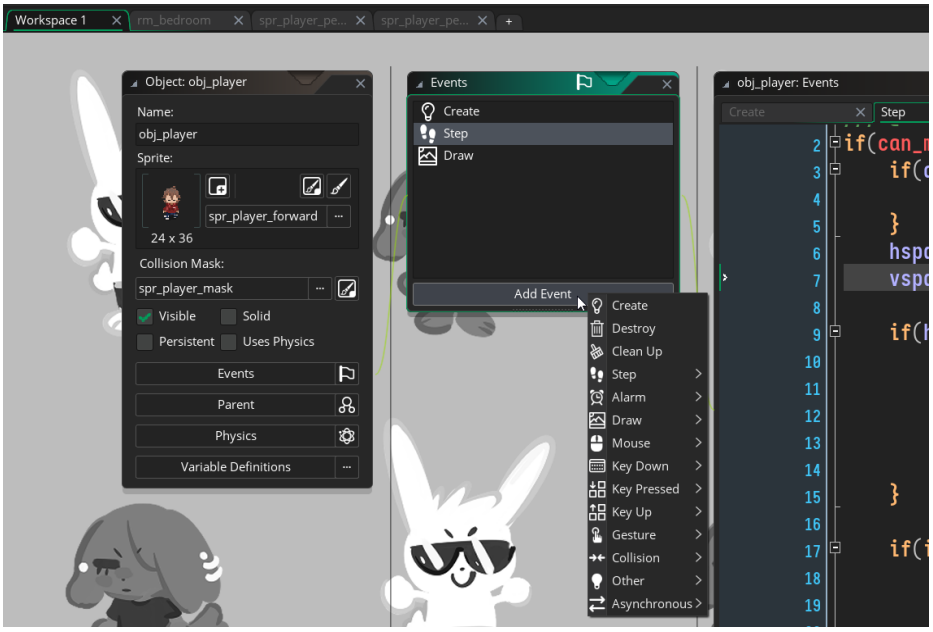
Et gamemaker projekt består af mange forskellige “Assets”
Assets inkludere:

- Lydfiler
- Sprites (animerede billeder)
- Rooms (kan også kaldes baner, levels)
- Objects (Objekter bliver placeret i rooms og eksekvere kode, samt tegner ting i spillet)
- Scripts (script filer kan indeholde en eller flere globale funktioner, som objekter kan benytte.

Der findes flere typer assets, Tilemaps, Fonts, Shaders, mm. Men de er ikke relevante for at forstå udviklings processen.

6.2 Objects

Det er vigtigt at forstå hvordan objekter fungere for bedre at forstå hvordan et gamemaker spil fungere. Når man laver et objekt, kan man efterfølgende placere en instans af det objekt i et Room, og når det room(level/bane) starter, vil det så påbegynde sine events.



Spiller objektets events

6.2.1 Object events

Objekter er delt op i forskellige events, der bestemmer hvornår forskellige stykker kode bliver eksekveret i objektet. Der findes mange, men for at gøre det enkelt vil jeg gå igennem de 4 mest anvendte.

Create event:

Create eventen indeholder kode som bliver eksekveret i det øjeblik at objektet bliver lavet. Så det vil blive eksekveret hvis man placere en ny instans ved hjælp af kode i løbet af spillet. Eller hvis man skifter til et room som indeholder den instans.

Den bliver brugt til at sette variabler som objektet skal bruge under sin levetid. Man kan også sætte en variabel til at være en function, så functionen ikke er global men kun tilhøre det objekt. Som så kan benyttes i step eventen f.eks.

Step event:

Step eventen indeholder kode som bliver eksekveret hver game frame.
Så i mit projekt 60 gange i sekundet.
Mit spiller objekts step event, tjekker f.eks løbende efter input og vil så bevæge spilleren.

Der findes 3 step events. Der bliver kørt i rækkefølge.

- step begin
- step
- step end

For kort at forklare hvorfor det kan være smart, så lad os sige var har placeret 2 objekt indstanser i et room som begge benytter alle 3 step events. Rækkefølgen af deres eksekvering vil være følgende:

1. Instans 1 - step begin
2. Instans 2 - step begin
3. Instans 1 - step
4. Instans 2 - step
5. Instans 1 - step end
6. Instans 2 - step end

Det er sjældent nødvendigt, men det er værd at vide at man har kontrol over rækkefølgen på den måde hvis man har rigtig mange indstanser der skal interagere med hinanden.

Draw event:

Draw eventen er meget lig med step eventen, i det at den bliver eksekveret hver game frame. Men den er lavet til at håndtere tegning af grafik.
Der findes funktioner i gamemaker som draw_sprite() draw_circle() draw_rectangle() mm. Som kun kan benyttes i draw eventsne.

Den har også 3 draw events.
Rækkefølgen fungere på samme måde som i step eventen.

Draw Gui:

Fungere ligesom draw eventen. Men som navnet hentyder er den lavet udelukkende til at tegne det grafiske user interface. Den vil altid biver tegnet oven på alt andet grafik. Og bliver i mit spil brugt til at tegne de forskellige menuer.

Post Draw:

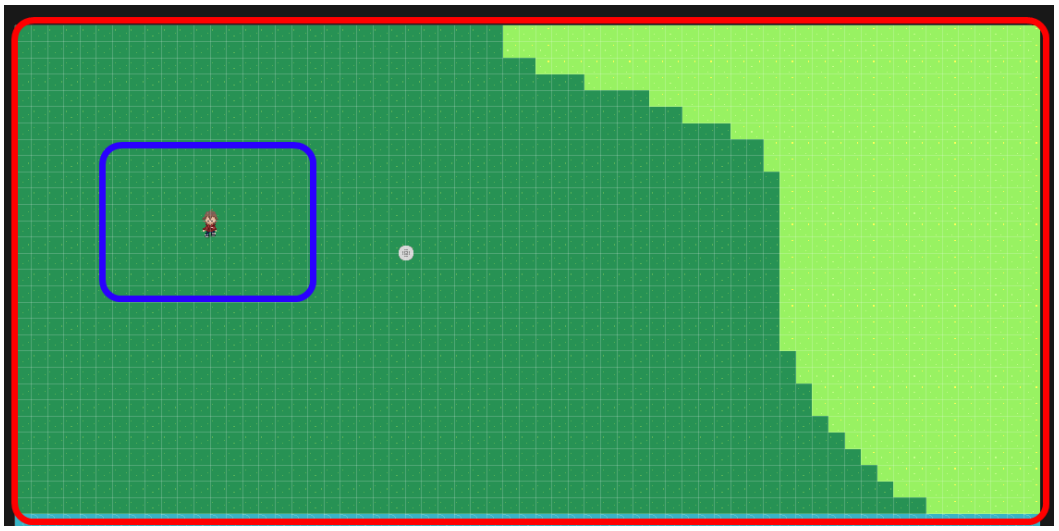
Dette er en særlig event der fungerer ligesom Draw Gui men bliver kørt lige før Draw gui starter, den bliver i mit spil brugt til at tegne indholdet fra det primære kamera til skærmen

7 Kamera system

Jeg vil først kort forklare hvordan Gamemaker som standard håndtere kamerae, og hvad et kamera betyder i denne her kontekst.

7.1 Hvad er et "kamera" i et spil?

Når man har et spil indelt i forskellige baner (rooms). Har alle baner en forskellig størrelse. Lad os sige en bane har en bredde og højde på 1000px x 450px
Men når vi spiller spillet skal hele banen ikke være synlig i vinduet hele tiden.
Vi vil gerne have at der bliver fokuseret på Spiller objektet, og man kun kan se 320px x 240px af banen ad gangen. Det kan man bruge et kamera til.



Room størrelse (rød) Kamera størrelse (Blå)

7.2 STANNcam

Som standard i gamemaker er der 8 kamera objekter man kan styre, ændre bredde, højde og position på. Før spil projektets start, havde jeg begyndt at lave et kamera styrings projekt til gamemaker, jeg delte offentligt på github, som jeg kaldte STANNcam[5], efter mit online alias.

7.2.1 STANNcam 1.0

Dens primære funktion var at styre skærmstørrelse og sikre at spillet blev opskaleret korrekt. Altså hvis man har et retro pixel spil, skal spillet opskalleres så det ikke er super småt på skærmen. Den gjorde det også nemt, at bestemme hvilket objekt kameret skulle følge, eller bevæge det til en ny position i en bane. Zoome ind og ud. Lave en ryste effekt mm. Dog kunne det kun håndtere et enkelt kamera objekt ad gangen.

[5] STANNcam 1.0 https://github.com/jack27121/STANNcam

7.2.2 STANNcam 1.8

I mit spil projekt fandt jeg ud af at det var nødvendigt at kunne anvende flere kamerae samtidig, for at opnå yderligere visuelle effekter. Derfor brugte jeg tid på at ændre systemet, så man løbende kan lave eller slette nye kamerae. Og så manipulere dem individuelt. Og nemt bestemme hvor på skærmen, det kameraet ser skal tegnes.

Efter opdateringerne delte jeg den nye version som et pre-release også på github. STANNcam 1.8[6] Og i løbet af spil projektet, har jeg så opdateret og finjusteret det yderligere. Efter projektets forløb vil jeg så også kunne gøre klar til at udgive v2.0 og skrive ordentligt dokumentation dertil.

7.2.3 Anvendelse af STANNcam

I version 1.0, siden der kun var et enkelt kamera i gang på ethvert givent tidspunkt, havde jeg sat det op til automatisk at instantiere sig selv i spil projektet. Og man kunne så benytte nogle globale funktioner til f.eks. At bevæge, zoome, ryste, ændre skærmopløsning mm. på det enkelte kamera. Nogle globale variabler var også sat til at bestemme skærmstørrelsen ved siden af.

I version 1.8 ændrede jeg hele strukturen, men den grundlæggende kode er dog den samme. Siden man har mulighed for mere end et enkelt kamera, skal man manuelt instantiere og sætte dem til variabler. Og funktionerne til at ændre deres indstillinger ligger så nu i variablerne. Herunder er et kode eksempel. Taget fra mit primære kamera objekt. Alt koden kan ses her
https://github.com/jack27121/yume-newkki/tree/master/objects/obj_camera

Create event:

```
stanncam_init(320,240,1920,1080);
//Instantiere kamera systemet
//Sætter globale variabler for spil oplysningen til 320 x 240
//Og den endelige skærmopløsning til 1920 x 1080

global.camera = new stanncam(0,0,global.game_w,global.game_h);
//Et nyt kamera bliver lavet og sat til en global variabel global.camera
//Den bliver lavet på position 0,0
//Og sætter bredden og højden til de nye globale variabler init funktionen lavede
```

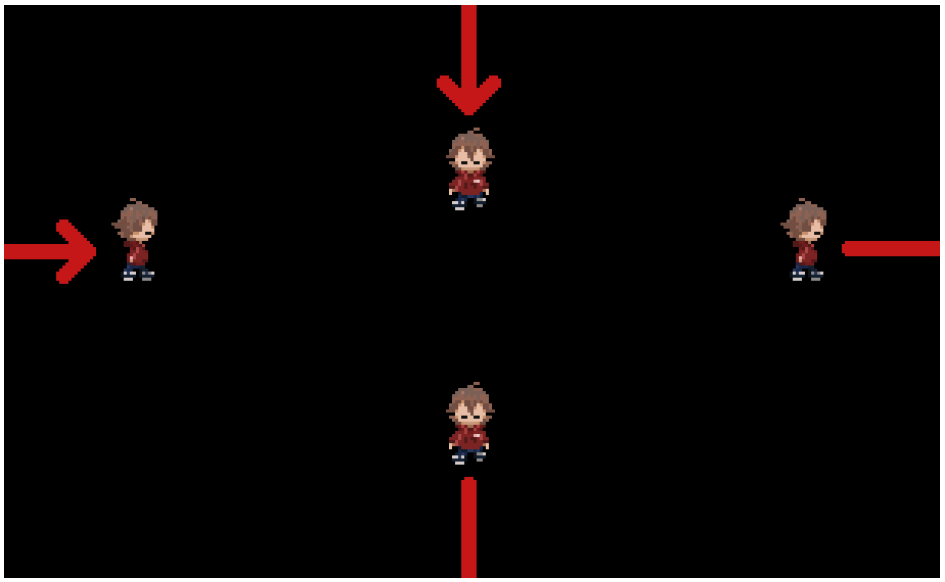
Post-Draw event:

```
global.camera.draw(0,0);
//Denne tegner hvad kameraet ser på skærmen. internt har jeg gjort så draw
//funktionen tegner i den rigtige oplysning til spil vinduet
```

[6] STANNcam 1.8 https://github.com/jack27121/STANNcam/releases/tag/v1.8.0

7.3 Room wrapping

Jeg nævnte at det blev nødvendigt at benytte mere end et enkelt kamera ad gangen i mit projekt. I *Yume Nikki*, som mit spil er meget inspireret af, er der mange baner, hvor hvis man går ud til højre, kommer man ind til venstre. Og det samme med op og ned.



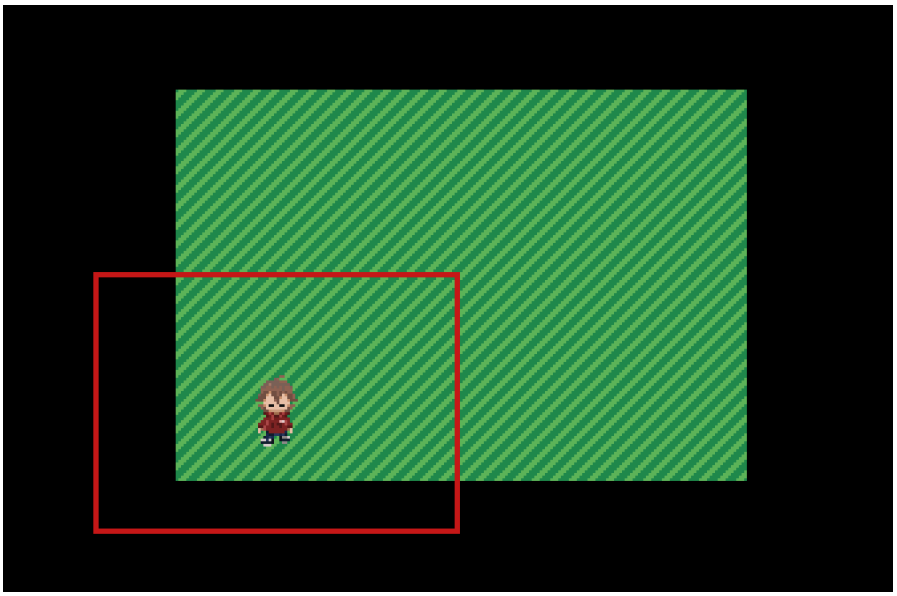
Spilleren går ud ad banen og kommer ind på den anden side

Det er nemt at teleportere spilleren til den anden side af banen når de kommer over bane grænsen. Herunder er noget pseudo-kode der demonstrere hvordan det kan gøres.

Spillerens Step event:

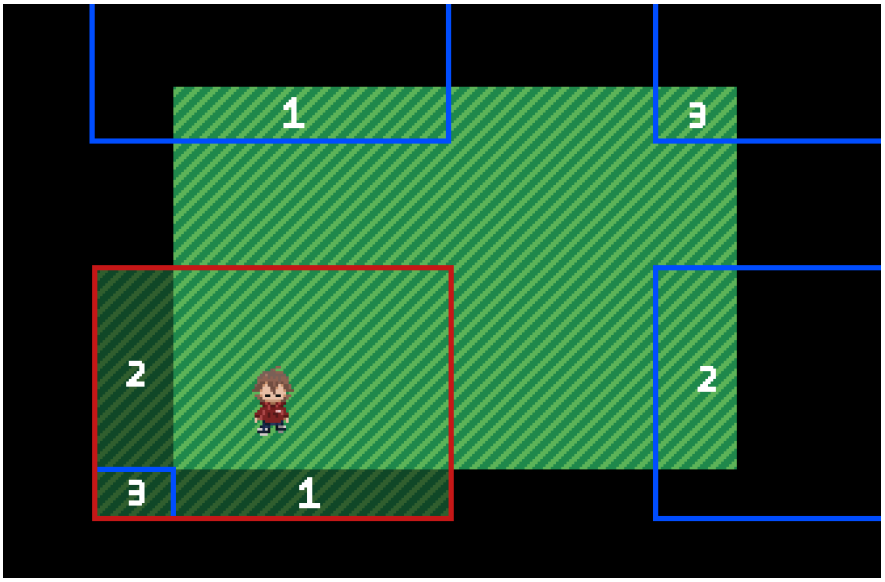
```
//room_width er breeden på det nuværende Room
//room_height er højden
if(x > room_width){
        x-= room_width;
};
if(x < 0){
        x+= room_width;
};
if(x > room_height){
        x-= room_height;
};
if(x < 0){
        x+= room_width;
};
```

Den svære del er at sørge for at kameraet ikke kan se grænsen af banen, før man bliver teleporteret. Som f.eks her



Den grønne kasse er banens areal. Den røde firkant er hvad kameraet ser.

Uden ekstra arbejde vil man kunne se sort hver gang man når grænsen. Og det er her ekstra kamerae kan løse problemet. Når det primære kamera når uden for grænsen kan vi aktivere ekstra kameraer i de andre ender af banen og rendere hvad de ser ovenpå det originale kamera



De blå firkanter er ekstra kameraer. Hvad de ser bliver renderet ovenpå det originale kamera.

8 State-machine (snowstate)

I mange dele a mit projekt benytter jeg state-machines, ved hjælp af extensionen Snowstate. Jeg vil her forklare hvad det er og hvordan det fungerer, så jeg nemmere kan forklare nogle af de andre spil elementer bagefter.

8.1 Hvad er en state machine?

En state machine er en måde hvorpå man kan dele sin kode op i forskellige “states” hvis man har et spiller object, kunne man f.eks have:

- “idle” state når spilleren står stille.
- “walking” state når spilleren går rundt.
- “attack” state når spilleren bruger et våben.

State machines er en rigtig god måde at styre de mange forskellige animationer en spiller kan have, og sætte specifikke variabler for forskellige states.
Man kunne f.eks ovenpå sin “walking” state lave en “running” state hvor koden er den samme, men hvor speed variablen er gjort hurtigere.

8.2 Hvad er Snowstate?

Snowstate[7] er en extension til gamemaker der gør det nemt at håndtere state-machines.

State-machines minder lidt om switch-cases. Og det kunne man også nemt bruge som en simpel state-machine. Men Snowstate giver ekstra funktionalitet.
Som “enter” og “leave” funktioner, samt nedarvning af states. Herunder viser jeg et pseudo kode eksempel på hvordan man kunne bruge state-machines til at håndtere sit spiller objekts animationer.

Den reflektere ikke mit rigtige spiller objekt. Det bliver beskrevet 10 Spiller objectet

[7] Snowstate https://github.com/sohomsahaun/SnowState

Spiller Create event:

```
State = new snowstate("idle");
//laver en ny state machine og sætter "idle" til at være start staten.
state.add("idle", {
	enter: function(){  //enter bliver kørt så snart denne state bliver valgt
		sprite_index = spr_player_idle;
		standing_still = true;
	},

	//step kan vi kalde i vores rigtige step event,
	//så koden køre hver frame idle staten er aktiv
	step: function(){
		//hvis hspd eller vspd ikke er 0 skifter vi til walking staten
		if(hspd != 0 || vspd != 0){
			state.change("walking");
		}
	},
	//leave bliver kørt når vi forlader denne state
	leave: function(){
		standing_still = false;
	}
});
state.add("walking", {
	enter: function(){
		sprite_index = spr_player_walking;
	},

	step: function(){
		//hvis hspd og vspd er 0 skifter vi til idle staten igen
		if(hspd == 0 && vspd == 0){
			state.change("idle");
		}
	},
})
```

Spiller step event:

```
hspd = (input_check("right") - input_check("left"));
vspd = (input_check("down") - input_check("up"));
//Her kalder vi state machines step event. Som både er defineret i idle og walking
//staten. Det vil sige at hvis idle er aktiv, er det dens step funktion der bliver kørt
state.step();
```

9 GUI Menuer

9.1 Menuer

De forskellige Gui menuer i mit spil benytter også en meget simpel state machine. De er alle sat op med en “selection” variabel der bliver pluset eller minuset, når man trykker op og ned på keyboardet/controlleren.

Alt menu koden kan ses her
https://github.com/jack27121/yume-newkki/blob/master/objects/obj_init/Create_0.gml

9.1.1 Idle

ingen kode bliver kørt, der er ingen menuer aktive.

9.1.2 Start_menu

Det er den første menu man ser når man starter spillet. Og er her man kan vælge at starte et nyt spil, loade et tidligere gemt spil. Eller lukke programmet igen.

9.1.3 new_save/load_save

Disse to menuer nedavre begge af parent_save menuen, da de begge skal vise alle de gemte saves.
I new_save menuen kan man lave nye saves eller overskrive et eksisterende.
Og i load_save menuen, kan man loade eller slette et eksisterende save.

9.1.4 Pause_menu

Pause menuen kan spilleren tilgå i spillet, og vælge imellem nogle andre menuer. Effects, og Settings. Samt at kunne lukke spillet igen.

9.1.5 Effects

Denne menu er til at vise og benytte alle de forskellige effects spilleren har samlet op i løbet af spillet.

9.1.6 Settings

Denne menu kan man ændre skærmstørrelse, om det skal være i fullscreen eller windowed mode, og lydniveau.

Mange af de forskellige sprites for spilleren

10 Spiller objektet

Spiller objektet er det man styrer igennem hele spillet, og som benytter de mange forskellige effekter man kan samle op. De mange forskellige effekter giver en stor udfordring for hvordan spiller objectet mest optimalt kan kodes. Mange effekter skal bruge noget af den samme kode, som f.eks. Movement og collision koden. Og samtidig skal nogle effekter have helt unikke egenskaber. F.eks pingvin effekten gør en i stand til at glide hurtigt rundt, og stick effekten ændrer ens kollisions boks.

Alt spiller objektets kode kan findes her
https://github.com/jack27121/yume-newkki/tree/master/objects/obj_player

10.1 Effect states

Til at hjælpe med kode optimeringen har jeg igen taget state machines i brug for spilleren. Spilleren er delt op i mange forskellige states delt ud på alle de forskellige effects i spillet. Og de fleste states er så nedarvet af andre så koden er genbrugt mest optimalt.

Input, altså at sætte horizontal og vertical hastighed, samt andre små basale ting, sker i step eventen seperat fra alle states. Og collission/movement funktionen bliver så kaldt i de forskellige states, så det er muligt at ændre det hvis en effekt er meget unik.

Herunder gennemgår jeg de forskellige effekter og de underliggende states der tilhøre på spilleren.

10.1.1 Normal

Normal er altså når ingen effekter er anvendt. Den indenholder de mest basale states, som mange af de andre effecters states nedarver af.
Statesne er:

- idle

  Når spilleren står helt stille, den tjekker løbende om hspd og vspd (horizontal og vertical input) ikke er 0, og vil så skifte til walking staten.
  Alle de andre effekter har deres egen idle state som er nedarvet af denne. Med den eneste forskel at de benytter deres egen unikke sprites.

- walking

  Når spilleren bevæger sig. Den køre løbende collision funktionen som benytter hspd og vspd. Som tjekker for kollisoner med omverdenen, og hvis muligt bevæger spilleren. Den tjekker også løbende om hspd og vspd er 0, og i så fald skifter tilbage til idle staten.
  De fleste andre effekter har også deres egen walking state og bliver nedarvet af denne.

```
state.add("walking", {
    enter: function(){
        animate = true;
        subimg = 0;
        idle_state = "idle";
    },
    step: function(){
        if(hspd == 0 && vspd == 0){
            state.change(idle_state);
        }
        collision(hspd,vspd); //movement and collision
        wake_check();
    }
});
```

- slap

  Man kan trykke på en knap for at vågne op igen fra en drøm. Det for spilleren til at daske sig selv i hovedet. De fleste andre effekter har også deres egen slap state og bliver nedarvet af denne. Når nogle effekter er aktive så som Nose effekten har man ingen arme. Og i det tilfælde er denne state så helt slået fra.

10.1.2 Stick

Stick effekten, forvandler spilleren og gør ham ligeså tynd som en pind. Og gør også spilleren en smule langsommere. Dog gør det også spillerens kollisions boks mindre, hvilket gør spilleren i stand til at krybe igennem meget smalle åbninger som findes nogle steder i drømme verdenen. Alle dens states er nedarvet fra Normal.

10.1.3 Penguin

Penguin effekten forvandler spilleren til en pingvin. Spilleren bliver en smule langsommere når de går rundt normalt. Men har her også mulighed for at lægge sig på maven for at glide rundt, meget hurtigere end hvis man gik normalt. Den nedarvre alle de basale states fra Normal, men har en tilføjelse der tjekker får input. Man kan trykke på en knap der så skifter til slide staten.

- Slide

  Penguin har sin egen ekstra unikke state, til at glide. Koden minder en smule om walking staten, men har ekstra kode for at få karaktereren til accelere og deaccelere når man bevæger sig.

```
step: function(){
        //slide_dir ændre gradvist retning. Og ændre subimg
        //på penguin slide spriten, så det ser ud som om den roterer
        slide_dir += angle_difference(dir,slide_dir) * 0.06;
        subimg = round(slide_dir / 30);

        //her bliver slide_hspd langsomt ændret til at matche
        //hvad end hspd er sat til. De giver en accelerations effekt
        slide_hspd = lerp(slide_hspd,hspd,0.04);
        slide_vspd = lerp(slide_vspd,vspd,0.04);

        //tjekker for input, og i så fald går ud af slide staten igen
        if(input_check_pressed("special")){
                state.change("penguin_idle");
        };
        collision(slide_hspd,slide_vspd);
},
```

10.1.4 TV

TV Effekten ændrer karakterens hoved til et TV. Og tv'et kan så tændes og slukkes ved at trykke på en knap. Den er på nuverende tidspunkt kun visuelt, men jeg vil senere give mere funktionalitet til. Som F.eks. At oplyse mørke rum, eller tiltrække NPC'er. Alle states er nedarvet af normal.

10.1.5 Nose

Nose effekten forvandler spilleren til en kæmpe næse. Den er også pt kun visuel. States er nedarvet af Normal

11 Gemme data

Alt user data bliver gemt i JSON filer lokalt på computern. JSON er nemt at skrive og læse fra, og gamemaker har funktioner der hurtigt kan håndtere dem.

Man vil være i stand til at finde filerne på *%localappdata%/yume_newkki*

De forskellige funktioner jeg bruger til at gemme, slette og loade, er her https://github.com/jack27121/yume-newkki/blob/master/scripts/save_load_functions/save_load_functions.gml

11.1 Saves

Saves. Altså en spillers gemte spil, hvor mange effekter de har opsamlet, tid der er spillet osv. De forskellige spil bliver gemt i en "saves" folder. Og navngivet efter hvilket nummer der blev valgt at gemme til.

save0.json :

```
{
  "time_played": 0.0,
  "money": 0.0,
  "Effects_enabled":
    [ true, false, true, false, false, false, false ]
}
```

Man kan se at de opsamlede effekter blot er et array af true/false.
Da ver effekt har et id, kan man tjekke positionen i arrayet for at finde ud af om en effekt skal være slået til eller fra.

11.2 Settings

Alle de ting der kan ændres i indstillings menuen.
Skærmstørrelse, om det skal være fuldskærm. Samt ændring af sproget

settings.json :

```
{
  "fullscreen": true,
  "resolution": 4,
  "stretching": false,
  "language": 0
}
```

Man kan se at resolution bare er en integer. Det er fordi de forskellige skærmopløsninger ligger som et array hardcoded inde i spillet, så integeren er brugt som et index til at hive den rigtige resolution ud af det array. Det samme er tilfældet med language.

12 Sprog (Lexicon)

Som del af settings menuen kan man ændre sproget af teksten der er vist i spillet. Det er blot en integer variabel, og inde i spillet er der så en liste af mulige sprog. Der er gjort brug af ekstensionen Lexicon.[8]

Måden det fungerer på, er at der udenfor spillet ligger forskellige Json filer til hver enkelt sprog der er ledigt. Her er et udkast fra den Engelske sprog fil. Local_en.json Som også kan findes her
https://github.com/jack27121/yume-newkki/blob/master/datafiles/local_en.json

```
{
        "language": "English",
        "locale": "en-US",
        "text": {
         "gui": {
                "menu": {
                        "new_game_name" : "New Game",
                        "new_game" : {
                                "text" : "Create new game",
                                "popup_confirm" : "Override game?",
                        },
```

[8]Lexicon https://github.com/tabularelf/lexicon

```
                    "settings_name" : "Settings",
                    "settings": {
                            "resolution": "Resolution",
                            "fullscreen": "Fullscreen",
                            "language": "Language",
                    }
            }
        },
        "effects": {
                "stick": {
                        "name": "Stick",
                    "description": "Very stick like"
                },
            "penguin": {
                    "name": "Penguin",
                    "description": "Known to slide around on it's belly"
                    },
        }
}
```

Man kan se at det er en masse keyvalue pairs.Inde i spillet, i de forskellige sprog filer vil Keyen være ens, mens valuen har vær sit sprog.

hvis jeg f.eks ville hente tektsen til *"new game"* menu punktet kan jeg i koden skrive

```
var text = lexicon_text("gui.menu.new_game_name");
```

Inde i settings menuen når man skifter sprog bruger jeg denne kode

```
var lang_array = lexicon_languages_get_array();
/// giver et array af alle de mulige sprog og locals,
/// i mit tilfælde [
///                  ["English","en_US"],
///                  ["Dansk","da_DK"],
///                  ["Russian","ru_RU"]
///                 ]

lexicon_language_set( lang_array[global.settings.language][0] );
///global.settings.language er integeren der er sat fra vores settings fil
///Her bliver language altså sat til "Dansk"
```

13 Projekt Logbog

Se Billag/logbog.pdf

14 Realiseret tidsplan

Se Billag/realiseret_tidsplan.pdf

Min estimerede tidsplan var ikke helt fyldt ud med tid, da jeg på det tidspunkt ikke var sikker på hvor langt tid mange ting ville tage, og jeg tænkte at jeg skulle bruge meget lang tid på at skrive disse rapporter, men i stedet lavede jeg de forskellige rapporter samtidig med at jeg udviklede.

15 Afgivelser

Nogle ting blev nedprioriteret i løbet af udviklingsprocessen og fjernet fra min kravspecifikation, til at give mere tid til at fikse små-fejl og færdiggøre rapporterne. Alle de elementer vil dog blive tilføjet senere efter eksamensforløbet, elementerne er beskrevet herunder.

15.1 Færre rum/verdener

En stor del af spillet handler om at udforske forskellige verdener, men at designe og tegne de verdener er ikke særlig kode tunge og blev af den grund ned-prioteret, i løbet af eksamensforløbet. Så i den version jeg viser er der kun 2 små eksempel verdener.

15.2 Gun effect

Denne effekt ville være mere involveret da den ville give spilleren en pistol som der kunne blive affyret for enten at ødelægge visse objekter eller dræbe NPC'er. Den blev nedprioriteret da den også kræver en del ekstra animationer og samtidig skulle have meget ekstra unik kode for at affyre skud. Jeg skulle i så fald også opdatere NPC koden så de kunne blive slået ihjel af den.

15.3 Items

Items ville fungerer lidt lig med effects. I det at de kan opsamles og så anvendes i en items menu. F.eks en nøgle til en dør. Eller et brev til en postkasse.
Menuen og noget af koden til at instantiere Items og dens menu blev faktisk lavet samtidig med effekt systemet blev lavet, men blev nedprioriteret da effekterne var vigtigere for mig at fokusere på.

16 Konklusion

Da jeg startede med dette projekt var min intention at opnå noget lig en version 0.1.0 Med mulighed for så at videreudvikle det efter, og løbende putte flere ideer ind. Det synes jeg at have opnået fint. Alle de basale teknologier er nu implementeret, så at lave nye verdener, effekter og eller scenaraier vil nu være meget lige til.

Mit mål er at få flere elementer, verdener og effekter ind, og så udgive spillet i en alpha version, muligvis på spil distributøren Steam, og delt gratis. Og så fortsat opdatere det løbende. Med held kan det på et tidspunkt få sin egen fan-kreds.

Derudover fik jeg igennem projektet løbende opdateret min Gamemaker ekstention STANNcam. Som jeg nu snart kan udgive en ny version af, og skrive ordentlig dokumentation til.

16.1 Problemformulering Spørgsmål besvaret

1. Hvordan kan man bedst muligt anvende forskellige effekter?
   Måden jeg fik optimeret koden til de mange forskellige effekter var ved brug af en state machine til spilleren. Spilleren blev opdelt i mange forskellige states, og de fleste af effekterne nedarver af de første for at genbruge en stor del af koden.
   Jeg gik mere i dybden med dette i 10.1 Effect states

2. Hvordan kan man gøre det nemt at have forskellige sprog i spillet?
   Jeg ville gerne gøre det nemt for mig selv senere at inkludere flere sprog, og til det benyttede jeg extensionen lexicon.
   Der gjorde mig i stand til at putte alt tekst i seperate sprog filer. Så hvis jeg vil inkluderer flere sprog senere, eller hvis der en dag i fremtiden er fans af spillet der selv vil oversætte det, kan de nemt tilføje deres egne sprog filer.
   Det blev forklaret i 12 Sprog (Lexicon)

3. Hvordan kan man have mulighed for at have baner som wrapper ind på hinanden seamless, altså hvis man går ud til højre kommer man ind til venstre?
   At få flyttet spilleren til den anden side af en bane var nemt. Men at sørge for at man ikke kunne se nogle huller var mere involveret. Og det var derfor jeg blev nødsaget til at opdatere mit kamera system STANNcam til at kunne håndtere flere kameraer. Når man kommer tæt på kanten af en bane, er der ekstra kamerae i den anden ende a banen, som så bliver rendered ovenpå hullet for at fylde det ud.
   Det blev beskrevet i 7.3 Room wrapping