ARCADA

Speldesign i Godot

Patrik Sigfrids

Examensarbete

Informationsteknik

2021

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	
Författare:	Patrik Sigfrids
Arbetets namn:	Speldesing i
Handledare (Arcada):	
Uppdragsgivare:	

Sammandrag:
Detta arbete går ut på att bygga ett 2D spel med Godot spelmotorn. Godot är en Spelmotor med öppen källkod publicerat under MIT-licensen. Första versionen av Godot kom ut 2014. Godot försöker erbjuda en integrerad utvecklingsmiljö. Den låter spelutvecklare laga spel från början till slut utan att behöva använda andra verktyg annat än för att laga innehåll till spelet som grafik och musik. Spelet som byggs är ett enkelt 2D spel som är sett uppifrån. Spelet går ut på att ta kontroll över ett rymdskepp och strida mot andra rymdskepp. Rymdskeppet styrs med tangentbordet och attacker sker med musen. Spelet är menat att spelas mot andra spelare. Spelets grafik består huvudsakligen av sprites med lite partikeleffekter för motorer och explosioner. Fysiken i spelet är förverkligad med hjälp av de i Godot inbyggda noderna KinematicBody2D, Area2D och CollisionShape 2D. Multiplayer använder Godots High-Level API. Multiplayer modulen fungera dock inte fullständigt.

Nyckelord:	
Sidantal:	
Språk:	
Datum för godkännande:	

INNEHÅLL

1 **INLEDNING** **3**

1.1 Bakgrund 3

1.2 Syfte och mål 3

1.3 Avgränsning 3

1.4 Metod 4

1.6 Struktur 4

2 GODOT **5**

3 SPELET **7**

3.1 Spelplan 8

3.2 Sprites för rymdskepp och missilerna 9

3.3 Visuella effekter för raketmotorerna på rymdskeppen och missilerna 10

3.4 Visuella effekter för missilexplosioner 10

3.5 Menysystem för att starta spelet. 10

4 LOGIK FÖR SPELET **11**

4.1 Spelplan 12

4.2 Rymdskeppet 12

4.2.1 Kod för skeppet *13*

4.2.2 HealthDisplay *18*

4.2.4 Missilen *19*

4.3 Partikeleffekter 20

4.3.1 Rymdskeppets motor *21*

4.3.2 Missilens motor *22*

4.3.3 Explosion *23*

4.4 Meny 24

4.4.3 Kod för menyn 24
4.5 Multiplayer 26
4.5.1 Global.dg 27
5 Resultat **31**
5.1 Vad som har uppnåtts 31
5.2 Vad som inte har uppnåtts 31
6 Diskussion och slutsatser **32**

Figurer

Figur 1 Bakgrund 9
Figur 2 Rymdskepp och missil 9
Figur 3 Sväng med sprite motor effekt mot med partikeleffekt 10
Figur 4 Menyn 11
Figur 5 Karta på strukturen 11
Figur 6 karta på noderna 13
Figur 7 Fysik metoden för skeppet 14
Figur 8 kod för indata lyssnare 14
Figur 9 kod för laddnings tiden 15
Figur 10 koordinatsystem 16
Figur 11kod för att låsa muspositionen 17
Figur 12 kof för att ta skada 18
Figur 13hälsostången 18
Figur 14 kod för uppdatering av hälsostången 19
Figur 15fysik metoden för missilen 20
Figur 16kod för att registrera tröffar 20
Figur 17partikeleffekt 22
Figur 18partikeleffekt flr missilen 22
Figur 19partikelffekt för explosionen 23
Figur 20kod för play knappen 25

Figur 21kod för star a server knappen 26
Figur 22kod för Join a game knappen 26
Figur 23kod för start a Game knappen 26
Figur 24metoden för att starta spelet 27
Figur 25kod för att lyssna på på nätverket 28
Figur 26metoderna som kallas från ready metoden 28
Figur 27metoden att registrera spelare 29
Figur 28metoden för att konfigurera spelet 29
Figur 29metoden för när spelet har konfigurerat 30
Figur 30metod för att stänga av pausen på spälet 30
Figur 31metod för att starta spelet 30

1 INLEDNING

I detta examensarbete behandlas processen för att utveckla ett spel i den gräsrotsfinansierade spelmotorn Godot. Med detta arbete som stöd kan läsaren sedan på egen hand klara av att bygga ett liknande spel. Godot är en spelmotor med öppen källkod utvecklad av Juan Linietsky och är finansierat via Patreon(”Patreon – Godot engine”, 2021). Motorn kom ut 2014(Linietsky, J., 2014) vilket betyder att den är mycket nyare än Unity-motorn som utkom 2005(Pekham. E, 2019) och Unreal-motorn som utkom 1998 (Thomsen, M., 2012). Detta betyder att det inte finns mycket dokumentation för hur man skall använda motorn när man bygger spel.

1.1 Bakgrund

Examensarbetet är inspirerat av faktumet att Godot är gratis och fritt tillgänglig programvara som är utvecklad av studenter. Godot spelmotorn kan ses som en konkurrent till de stora aktörerna, dvs Unity och Unreal, men frågan är hur kraftfullt den är och vilka möjligheter den bjuder på för spelutvecklare. Godot är också en relativt ny spelmotor vilket betyder att det inte finns så mycket dokumentation för motorn. Därför är inspirationen bakom detta arbete att undersöka lite i vad kan denna spelmotor göra. 1.2 Syfte och mål

1.2 Syfte och mål

Syftet med detta arbete är att ge en översikt över hur Godot motorn fungerar och dess diverse moduler. Arbetet utreder hur motorn används när man bygger spel och att skapa en dokumentation för spelutveckling med Godot.

1.3 Avgränsning

Projektet avgränsas till ett multiplayer spel. Alltså spelet är menat at spelas mot andra spelare, som de spelar på sina datorer. Godot motorn har en inbyggd modul för att möjliggöra multiplayer. Orsaken för detta att arbetet som skulle måsta göras för att

programmera virtuella spelare skulle ta lång tid och vara mycket avancerat. Fokus med projektet är också på själva spelet. Vilket betyder att det kommer att finnas områden som till exempel användargränssnittet som kunde förbättras som i det här skedet är bara en skelettversion som finns där för att spelet skall fungera.

1.4 Metod

Föra att bygga ett spel behöver man en utvecklingsmiljö, grafiska resurser för att bygga vad du ser på skärmen, och ett programmeringsspråk du använder för att få allting att fungera. Som utvecklingsmiljö andvänds Godot-motorn. Godot-motorn använder sig av ett eget programmeringsspråk som heter GC Script.

Språket är utvecklat enkelt för Godot-motorn och är baserat på Python programmeringsspråket. Detta språk är valt för att få spelet att vara så Godot-baserat som möjligt. För att bygga resurser för spelet, så som grafiken för själva skeppen, projektiler och övriga effekter, används en gratis sprite creator på internet som heter Piskel (piskelapp.com**)**. Denna sida blev vald för att den är gratis och ger möjlighet till allting som behövs för att förverkliga allt det grafiska för spelet.

Spelet som utvecklas är ett science fiction-krigsspel. Spelet är ett ”multiplayer” spel vilket betyder att spelet är tänkt att spelas av mer än en person. I detta spel är spelarantalet begränsat till 2 – 10 personer. Spelet utspelas i rymden och går ut på att varje spelare tar i besittning sitt eget rymdskepp. Varje spelare skall sedan styra skeppet och strida mot andra spelare. Varje spelare hör till ett lag om maximalt 5 personer. I spelet finns två lag, så totala antalet spelare blir sålunda maximalt 10. De två lagen krigar mot varandra.

1.6 Struktur

Resten av examensarbete är strukturerat enligt följande: En översikt på Godot motorn ges is kapitel 2. Spelet som byggts som grund för examensarbeteges i kapitel 3. Processen för att

bygga spelet ges i kapitel 4. Resultaten för Praktiska arbetet ges i kapitel 5. Slutsatser dras i kapitel 6 med kapitel 7 som källförteckning.

2 GODOT

Godot spelmotorn är en spelmotor ursprungligen utveklad av Juan Linietsky pch Ariel Manzur. Motorn är opensource och och är puiblicerad under MIT-licensen (Berman, D,. 2020). Godot försöker erbjuda en integrerad spelutvecklings miljö. Den låter spelutvecklare laga spel från början utan att behöva andra verktyg annat än för att laga innehåll till spelet (konst, musik etc). Godot motorns arkitektur är byggt runt konceptet om att vara ett träd med noder. Noderna organiseras inom scener som sedan är återanvändbara, instansbara, ärftliga och nestbara grupper av noder. Alla resurser i spelet (skript och och grafiskt innehåll) sparas som en fil i stället för i en databas. Detta sätt att spara informationen är menat för att underlätta kollaboration mellan grupper.

Spel lagade med Godot kan programmeras med C++, C#, Rust, Nim, eller D. Godot har också sitt eget inbygga programmeringsspråk som heter Gdscript som är mycket liknande pyhton. Till skillnad från pyhton så har GDScript stängare variabler och är optimiserad för Godots scenbaserade arkitektur. Motorn har också stöd för visuell programmering via den inbyggda VisualScript, ett likvärdigt visuellt programmeringsspråk till GDScript. (”Godot engine – features”, 2021)

Godots grafikmotor använder OpenGL ES 3.0 för alla platformer som stöder det, i annat fall använder den OpenGL ES 2.0. OpenGL är en grafik API som spel och andra grafiska applikationer använder för att ge instruktioner åt grafikkortet om vad den skall rita på skärmen. Fördelen med OpenGL är att den fungerar på olika operativ system och med olika grafik kort. (”Godot engine – features”, 2021)

Godot inkluderar en 2D grafikmotor skilt från dess 3D grafikmotor. 2D grafikmotorn stöder följande funktioner: ljus, skuggor, skuggare, tegeluppsättningar, parallaxrullning, polugoner, animationer, fysik och partiklar (”Godot engine – features”, 2021)

Godot innehåller ett animations system med ett visuelt användargränssnitt (GUI) för skelett animation, animations träd, morfning och realtids scener. Nästan alla variabler definierade i ett spel kan animeras. Alltså man kan justera deras värden med tiden för att få en visuell effekt(”Godot engine – features”, 2021)

Ytterligare funktioner:

- Prestanda analys grafer
- Ljus bakning
- Multitrådsteknik
- Ett system för instickningsmoduler
- Videouppspelning
- Audio uppspelning
- Pertickel system
- Ett system for att exportera, importera och komprimera texturer
- Navmesh (en data struktur och algoritm som används för AI navigation)
- GUI (visuelltanvändargränssnitt)
- Stöd för tangentbord, mus, spelkontroll och pekskärm

Godot motorn är nämnd efter Samuel becketts teaterpjäs I Väntan på Godot (Linietsky, J,. 2014), Godot var valt för att det representerar den oändliga önskan att skapa bara mera och mera nya funktioner in i motorn, vilket får den att komma närmare och närmare till fullständig produkt, men den uppnår aldrig målet. Källkoden för Godot motorn publicerades allmänt på internet via publika webb utveklings hotellet Github under MIT-licensen. Version 1.0 av motorn publicerades 14 december 2014 och var den första stabila versionen av Godot motorn. Den nyaste versionen av Godot motorn är för tillfället 3.2. Godot motorn har fått finansiellt stöd av Mozilla och Epic games. (Etcheverry I.R., 2017)

3 SPELET

Spelet jag bygger här är ett 5 mot 5 multiplayer-spel där varje spelare kontrollerar ett rymdskepp. Spelarna skall sen strida mot varandra tills någondera av lagena har dött.

Spelet är ett uppifrån sett 2D spel där rymdskeppen flyger omkring en begränsad spelplan. På spelplanen finns ingenting förutom rymdskeppen. Alla stjärnor, planeter och annat som syns finns bara grafiskt i bakgrunden. Rymdskeppet du styr kommer alltid att ligga mitt i rutan. Alltså rutan kommer alltid att följa med ditt rymdskepp.

Rymdskeppen styrs med tangentbordet med knapparna W, A, S, D. W ökar på gasen. S sänker på gasen. A svänger rymdskeppets roder åt vänster vilket får rymdskeppet att svänga till vänster medan D svänger rodret till höger och får rymdskeppet att svänga till höger. När du svänger rymdskeppet så måste du hålla knappen i bottnet för att öka hur skarpt rymdskeppet svänger.

Tillika som du styr rymdskeppet kommer du att använda musen för att sikta dina vapen. När du flyger omkring i rymden kan du skjuta missiler. Missilerna kommer att följa din muspekare ända tills den passerats. Missilerna skjuts genom att trycka på musknappen. Därmed kommer rymdskeppen att ha kanoner på sidorna som skjuter missiler som flyger rakt efter att ha blivit avfyrade. Dessa missiler skjuts genom att ha muspekaren på endera sida av skeppet och trycka på mellanslagstangenten som avfyrar iväg fem stycken missiler åt det hållet.

När missilerna kommer tillräckligt nära ett fientligt skepp kommer missilerna att explodera vilket sedan skär ner en del av fientliga skeppets hälsa. Rymdskeppen kommer att ha en hälsoindikator visualiserat med en hälsostång. Desto större stången är, desto bättre är hälsan på ditt skepp. När hälsostången tar slut sprängs ditt rymdskepp.

Spelet är inspirerat av gamla science fiction bokserien Honorverse skriven av David Weber och Eric Flint av var första boken var publicerad i 1992 (Weber, D. 1994). Serien är en

såkallad militaristisk science fiction serie och behandlar mycket strider i rymden mellan rymdskepp. Serien har också mycket med fanart omkring internetet som det är lätt att basera konst i spelet utav. Senaste boken i serien var publicerad 2018.

Så för att förverkliga det här måste spelet brytas ner till funktioner. Funktionerna som behövs för att bygga spelet är:

- En spelplan
- En bakgrund för spelplanen
- Sprites för rymdskeppen och missilerna
- Visuella effekter för raketmotorerna på rymdskeppen och missilerna
- Visuella effekter för missilexplosioner och rymdskepps explosioner
- Logik för att läsa indata och uppdatera rymdskeppets gas och roderställning
- Logik för att flytta rymskeppet enligt gas och roderställning
- Logik för att generera missiler på basen av indata
- Logik för att uppdatera missilernas position enligt dess riktning och hastighet
- Logik för att justera missilernas riktning enligt indata
- Logik för att kolla om missilen har träffat ett fientligt rymdskepp och spränga den
- Logik för rymdskepp tar skada av missiler som sprängs i närheten
- Menysystem för att starta spelet samt logik för menyn.
- Logik för att multiplayer funktionaliteten

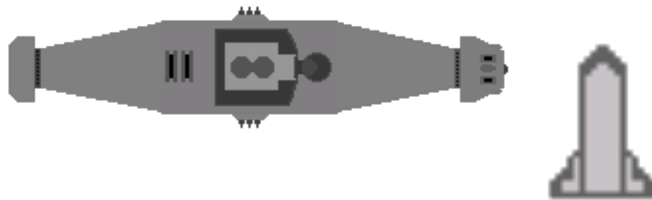
3.1 Spelplan

En Spelplan är området där spelaren kan röra sig omkring. Spelplanens storlek är 5000 x 5000 pixel. Om spelaren kör fast med kanten av spelplanen kommer spelarens rymdskepp att sprängas. Kanten på spelplanen är klart synlig en röd kant. Spelplanen själv kommer att ha en pixelkonst rymd bakgrund (se figur 1). Denna bakgrund kommer att upprepa så länge den är inom själva spelplanen.

Figur 1 Bakgrund

3.2 Sprites för rymdskepp och missilerna

Rymdskeppen och missilerna behöver grafik. Grafiken för rymdskeppen och missilerna är byggda med sprites. En Sprite är en 2 dimensionell bitmap som är integrerad in i en scen (i det här fallet spelplanen). Det är egentligen mindre bilder som sen kan bli manipulerade av själva spelet. En spelpjäs i digital form så att säga.

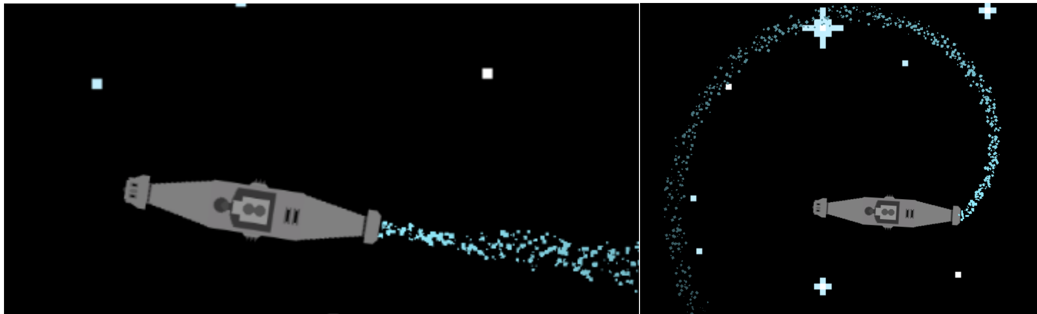


Figur 2 Rymdskepp och missil

Sprites är lagade med ett skilt verktyg. Verktyget som används heter piskelapp.com (”Piskelapp” 2021). piskelapp.com är en webbapplikation för att laga sprites eller annan pixel konst.

3.3 Visuella effekter för raketmotorerna på rymdskeppen och missilerna

Rymdskeppen och missilerna har en raketmotoreffekt som skall följa efter missilerna och rymdskeppen. Dessa är gjorda med en partikeleffekt. Orsaken varför effekten är gjort på det här sättet är att om den skulle vara byggd som en skild sprite så skulle effekten vara konstig när du svänger rymdskeppet. För att det skall se verkligare ut skall ju effekten följa efter där rymdskeppen och missilerna har varit, in nödvändigt vis rakt bakom själva spriten.



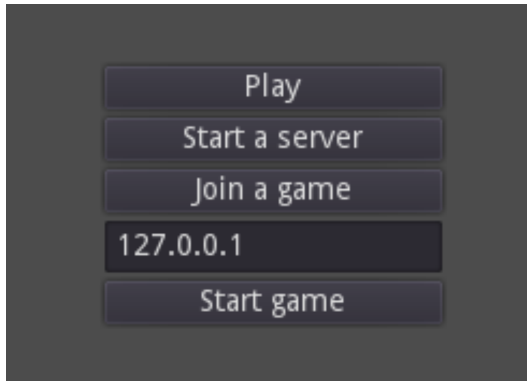
Figur 3 Sväng med sprite motor effekt mot med partikeleffekt

3.4 Visuella effekter för missilexplosioner

Missil explosioner förverkligas också med partikeleffekter. I ställe för en blå nyans så har explosionen en orange nyans. O partiklarna flyger i alla riktningar för att ge en bra explosions visualisering.

3.5 Menysystem för att starta spelet.

När du startar själva spelet skall du inte direkt vara inne på spelplanen och styra ditt rymdskepp utan du skall börja i en meny där du har sedan en knapp för att starta spelet själv. Den här delen är lämnad avsiktligt mycket grundläggande. Den har en bakgrund och lite färger bara för att själva spelet skall fungera.

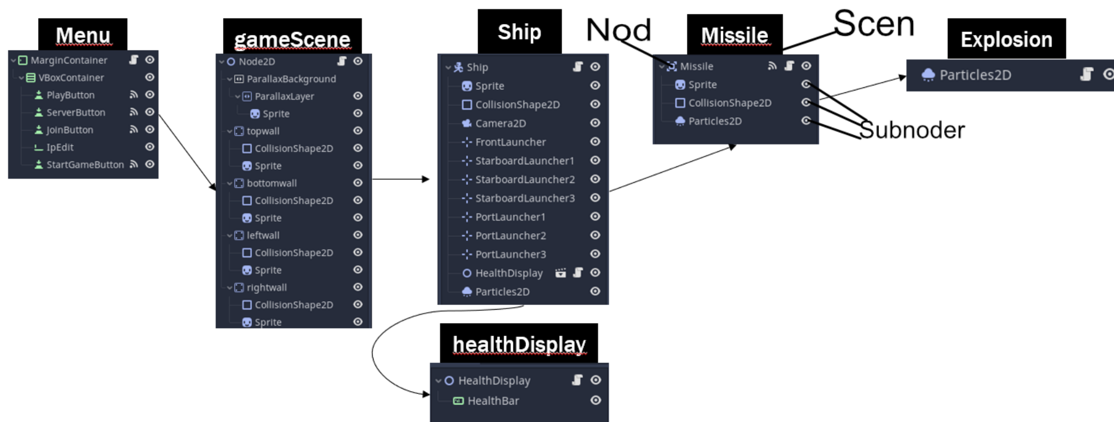


Figur 4 Menyn

4 LOGIK FÖR SPELET

Godot bygger på en hierarki av noder. En nod är byggblocken som man använder för att bygga upp spelet. Noderna själva kan ha subnoder. Till exempel så finns det en nod som heter Node2D och den representar någonting som finns i en 2 dimensionel värld. Man kan ge en Node2D en subnod av typen Sprite. Den ger noden dess grafik.

Noderna organiseras in i scener. När man kör spelet körs då alltid en ny scen som är uppbyggd av alla noder kopplade till denna scen. Där på kan man bygga upp skilda scener som kan laddas in i andra scener. Det här är ett bra sätt att organisera sitt spel.



Figur 5 Karta på strukturen

4.1 Spelplan

Spelplanen är den scenen som används när spelet startas. Dens rotnod är en Node2D nod som innehåller följande subnoder.

En **ParallaxBackground** nod. En bakgrundsnod som använder sig av en eller flera ParallaxLayer subnoder för att producera en parallaxeffekt där bakgrunder rör sig med en annan hastighet än saker i förgrunden. Det här spelet använder sig bara av en **ParallaxLayer**. Den har en **Sptire** subnod som ger den bakgrundsbilden.

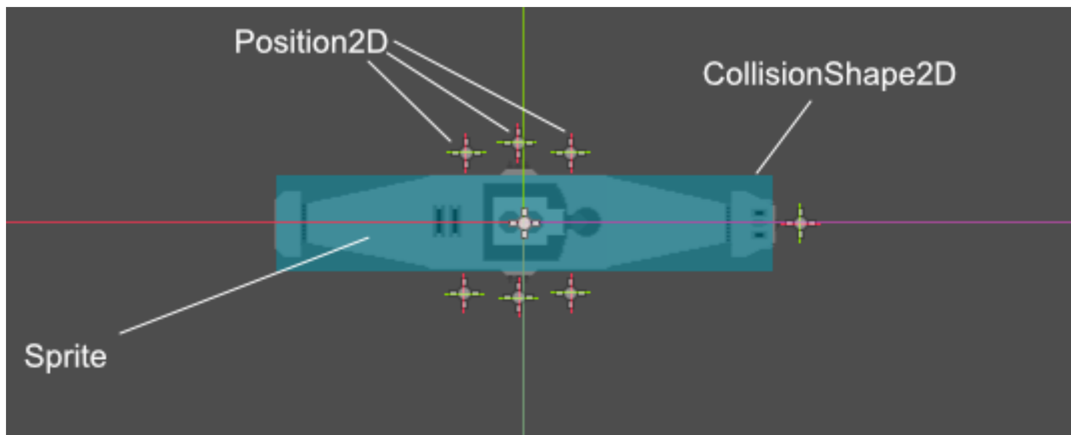
Fyra stycken **StaticBody2D** noder. En StaticBody2D är ett statiskt objekt i spelvärlden. Dynamiska objekt kan krocka med en StaticBody2D nod men noden själv kan inte röra på sig. Dessa fyra noder utgör gränserna för spelplanen.
Varje nod har en **Collisionshape2D** subnod som utgör området man kan krocka med. De har också en Sprite nod för att synas på spelplanen. Varje nod är 5000x60 pixel stor och utplacerade så de utformar en kvadrat. Det här gör spelplanens storlek till 5000x5000 pixel. Spriten använder som bild en 1x1 pixel stor röd bild som upprepas. Denna scen innehåller ennu en skild scen som är rymdskeppet.

4.2 Rymdskeppet

Rymdskeppet är själva spelpjäsen du tar kontroll över för att röra dig om kring i spelvärlden. Den är uppbyggd som en skild scen som innehåller följande noder.
En **Sprite** nod som ger skeppet sitt utseende. En **CollisionShape2D** nod för att ge själva skeppet en fysisk form i spelvärlden. Det är den här formen som krockar med andra fysiska former i spelvärlden. Denna nods storlek är 124x24 pixel stor och täcker hela skeppet.

En **Camera2D** nod, den är en kameranod för 2D scener som kameran sitter fast i och får rutan att följa med denna scen. Så här hänger spelaren med sin egen spelpjäs.

Sju stycken **Position2D** noder. Denna sorts nod representerar en punkt på ett 2D plan. De här sju noderna fungerar som vapen på skeppet. Skeppet har alltså sju stycken missil kanoner. En framåt, tre stycken mot styrbord och tre stycken mot babord. Dessa noder skall skapa missilerna som skall styras mot fienden. Noderna ligger en aning utanför själva skeppet för att de inte skall kollidera med skeppet själv när de skapas.



Figur 6 Karta på noderna

4.2.1 Kod för skeppet

Här kommer logiken för hur funktionaliteten för skeppet är programmerat.

4.2.1.1 Rörelse

Först måste skeppet ha funktionalitet för att röra på sig.

Skeppet har följande konstanter för rörelse definierade:

- Speed = 50. Bashastighet för skeppet uttryckt i pixel.
- Rotation_speed = 0.5. Basrotationshastighet för skeppet uttryckt i radianer.
- Max_rotation = 3. Högsta rotationshastigheten är 3 x Rotation_speed.
- Min_rotation = -3. Högsta rotationshastigheten för rotation i motsols riktning. -3 x Rotation_speed
- Max_burn = 3. Högsta hastigheten frammåt. 3 x Speed.
- Min_burn = -3. Högstqa hastigheten bakåt. -1 x Speed

Skeppet har följande variabler för rörelse definierade:

- Rotation_dir = 0. Hur mycket skeppet håller på att rotera för tillfället.

- Burn = 0. Hur mycket skeppet rör sig frammåt eller bakåt.

Node2D har en metod _physics_process som kör 60 gånger i sekunden. Denna funktion hanterar all fysik i spelet. Den fungerar genom att kalla på get_input metoden. Sedan kalkylerar den en ny Vector2 variable som beskriver skeppets hastighet på basen av två komponenter. Den första komponenten är hur mycket framåt skeppet skall röra sig på basen av Speed x Burn x Delta. Den andra komponenten är hur mycket skeppet skall rotera på basen av Rotation_dir x Rotation_speed x Delta. Delta är hur mycket tid som har gåt sen senaste uppdateringen. Den använder sedan hastigheten för att uppdatera skeppets position med metoden move_and_slide.

```
func _physics_process(delta):
    get_input()
    velocity = Vector2(speed * burn, 0).rotated(rotation)
    rotation += rotation_dir * rotation_speed * delta
    velocity = move_and_slide(velocity)
```

Figur 7 Fysik metoden för skeppet

Get_input metoden är den som läser om spelaren har tryck på någon knapp. Den fungerar genom att läsa all indata som spelaren gör och sedan mäter den om spelaren har tryckt på en viss knapp. Om spelaren har tryckt på knappen W så ökar den Burn variabeln med 1. Om burn variabeln redan är samma som Max_burn variablen så gör den ingenting. Samma gäller för knappen S som sänker på variablen med 1 ända till Burn variabeln ända tills den är samma som Min_burn variabeln. Samma gäller också för roteringen. Knappen D ökar på Rotation_dir med 1 till rotation_dir är samma som Max_rotation medan knappen A minskar på rotation_dir ända tills den är samma som min_rotation.

```
func get_input():
    if Input.is_action_just_pressed("right"):
        rotation_dir += 1
        if rotation_dir > max_rotation:
            rotation_dir = max_rotation
    if Input.is_action_just_pressed("left"):
        rotation_dir -= 1
        if rotation_dir < min_rotation:
            rotation_dir = min_rotation
```

```
if Input.is_action_just_pressed("down"):
    burn -= 1
    if burn < min_burn:
        burn = min_burn
if Input.is_action_just_pressed("up"):
    burn += 1
    if burn > max_burn:
        burn = max_burn
```

Figur 8 kod för indata lyssnare

4.2.1.2 Skytt

Sedan måste rymdskeppet ha funktionalitet för att skjuta missiler. För den finns det bara en konstant definierad:

- Reload_timer = 5. Hur länge det tar för att ladda en ny missil.

Och tre variabler:

- Front_reload = 0. Hur länge det är kvar innan framvapnet kan skjuta igen.
- starboard_reload = 0. Hur länge det är kvar innan styrborvapnet kan skjuta igen.
- Front_reload = 0. Hur länge det är kvar innan babordvapnet kan skjuta igen.

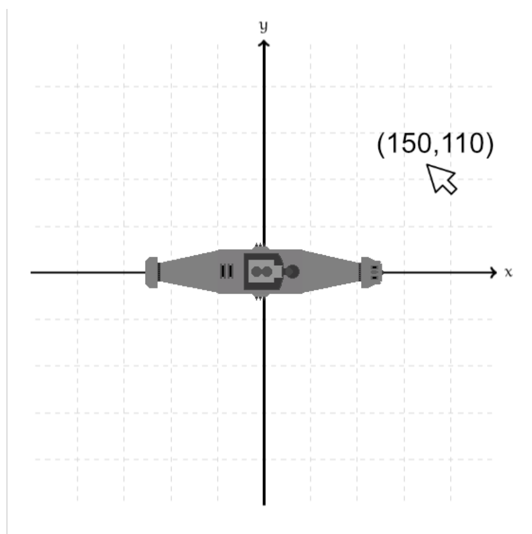
Orsaken varför dessa tider är delade upp i tre olika timers är för att skeppet har 3 olika positioner den kan skjuta ifrån. Ett vapen framåt, tre vapen mot styrbord och tre mot babord.

Sedan måste _physics_process updateras med logik för att beräkna det här. Den är byggd med en if kontroll. Alltså om Front_reload är större än noll så skall delta subtraheras från Front_reload. Samma logik för styrbord och babord.

```
if front_reload > 0:
    front_reload -= delta
if starboard_reload > 0:
    starboard_reload -= delta
if port_reload > 0:
    port_reload -= delta
```

Figur 9 kod för laddnings tiden

För att förverkliga logiken för att skjuta måste det definieras en ny metod. Shoot(). Först avläses muspekarens position i ett lokalt kordinatsystem med Godot funktionen ge_local_mouse_position. Den ger muspekar positionen i ett koordinatsystem centrerat på skeppet med x axeln pekande framåt och y axeln pekande mot babord.



Figur 10 koordinatsystem

Bilden ovan är ett exempel på ett koordinatsystem lokalt till rymdskeppet. Muspekarens position är (150,110). Sedan definieras in en if kontroll i metoden som kollar om muspekarens x-koordinat är åtminstone 100 pixel och att x-koordinaten är större än absoluta värdet för y-koordinaten och att det inte finns någon tid kvar i Front_reload. X-koordinaten måste vara större än 100 för att annars kunde man skjuta vapnet genom att trycka på skeppet och inte framför skeppet. Musepekarens x-koordinat måste vara större än det absoluta värdet av y-koordinaten för att fram vapnet skall skjuta endast om muspekaren är mera framför rymdskeppet än till styrbord eller babord om rymdskeppet. Om alla dessa kriterier fylls så avfyrar framvapnet. I exemplet ovan så skulle en missil skjutas framåt för att x-koordinaten (150) är större än 100 och y-koordinaten (110) är mindre än x-koordinaten.

När vapnet avfyrar updateras Front_reload variabeln med konstanten Reload_timer. Sen skapar den en ny missil med Missile.instance() metoden varefter den använder owner.add_shild metoden för att lägga till missilen den scen som rymdskeppet hör till. Till sist sätts missilens transform till framvapnets globala transform. Transform är en vektor som beskriver missilens position och rotation. Det gör att missilen placeras där var framvapnet befinner sig.

```
if mousePos.x > 100 && mousePos.x > abs(mousePos.y) && front_reload <= 0:
    front_reload = reload_timer
    var frontMissile = Missile.instance()
    owner.add_child(frontMissile)
    frontMissile.transform = $FrontLauncher.global_transform
```

Figur 11kod för att låsa muspositionen

Styrbord och babords vapnen har nästan samma logik men muspekarens koordinater måste var olika. För styrbord måste muspekarens y-koordinat vara större än 30 pixel eftersom missilen måste skapas 30 pixel till styrbord om rymdskeppet annars skulle man kunna skjuta genom att trycka på rymdskeppet. Muspekarens absoluta y-koordinat måste i detta fall var a större än muspekarens absoluta x-koordinat och Starboard_reload måste igen vara mindre eller lika med 0. Igen när alla dessa kriterier fylls så avfyrar alla 3 styrbordsvapnen, dvs. starboard_reload updateras med konstanten reload_timer och den skapar tre missiler på styrborbords sidan. De tre missilerna placeras på av de tre vapenpositionerare som finns styrbords och avfyras.

För att babords vapnen skall avfyras måste muspekarens y-koordinater vara mindre än en -30 pixel och muspekarens absoluta y-koordinat måste vara större än muspekarens absoluta x-koordinat. Och variabeln port_reload måste vara mindre eller lika med 0.

4.2.1.3 Skada

Skeppet har också en metod takeDamage() för att rymdskeppet skall kunna ta skada när den träffa s av en missil. För att förverkliga det har rymdskeppet ännu en konstant:

- Max_ health = 100

Och en variabel:

- Health = 100

Metoden takeDamage tar ett argument Damage som är hur mycket skeppet har tagit skada. Sedan subtraheras Damage värdet från Health. Om Health är mindre eller lika med 0 exploderar skeppet. Skeppet försvinner sedan med funktionen queue_free().

```
func takeDamage(damage):
	health -= damage
	$HealthDisplay.update_healthbar(health)
	if health <= 0:
		queue_free()
```

Figur 12 kof för att ta skada

4.2.2 HealthDisplay

För att visa åt spelaren att rymdskeppet har tagit skada så har skeppet en subscen HealthDisplay. Denna subsen är en Node2D som har en subnod TextureProgress som heter HealthBar. En TexureProgress nod är en nod som grafiskt visualiserar ett procentvärde från 0 till 100.

4.2.2.1 Kod för HealthDisplay

HealthDisplay har 3 variabler:

- Bar_red = preload("res://barHorizontal_red.png")
- Bar_green = preload("res://barHorizontal_green.png")
- Bar_yellow = preload("res://barHorizontal_yellow.png")

Dessa variabler tar bilder in som data och ändrar dessa bilder beroende på hur mycket skada dit rymdskepp har tagit.

Figur 13hälsostången

HealthDisplay noden har en metod _ready() som kör när noden skapas. Den gömmer HealthDisplay noden. Sedan sätter den HeatlhBar nodens max_value variabel till rymdskeppets max_health konstant. Noden tar den konstanten för att den skall veta vilket värde representerar hundra procent.

Metoden _process för HealthDisplay sätter global_rotation för HealthDisplay scenen till 0. _process metoden är en metod som kör varje bild. Global_rotation sätts till noll för att HealthDisplay scenen inte skall rotera med rymdsskeppet.

Till sist har HealtDisplay scenen en metod som heter update_healthbar. Den här metoden används av koden för rymdskeppet för att uppdatera HealthDisplay scenen när rymdskeppet tar skada. Som argument tar denna metod hur mycket Health som rymdskeppet har kvar. Om den har mera än 70 % kvar så använder den gröna bilden. Om har mellan 70 % och 35 % kvar så använder den gula bilden. Om rymdskeppet har mindre än 35 % kvar använder den röda bilden. Inne i metoden används det en metod som heter show(). Den får HealthDisplay scenen att visas bara om du har under 100 % Health kvar. Förverkligad igen med If kontroller.

```
func update_healthbar(value):
	healthbar.texture_progress = bar_green
	if value < healthbar.max_value * 0.7:
		healthbar.texture_progress = bar_yellow
	if value < healthbar.max_value * 0.35:
		healthbar.texture_progress = bar_red
	if value < healthbar.max_value:
		show()
	healthbar.value = value
```

Figur 14 kod för uppdatering av hälsostången

4.2.4 Missilen

För att kunna orsaka skada måste skeppen ha projektiler att skjuta, missiler. Dom är uppbyggda som en egen subscen. Scenen är av nodtypen Area2D. Area2D är en nod som används för att kolla om andra noder har kolliderat med denna nod. Den har två subnoder. En spritenod för grafiken för missilen och en CollisionShape2D nod för att kolla vilken form den har kolliderat med.

4.2.4.1 kod för missilen

Missile har tre konstanter:

- Speed = 500, hastigheten på missilen
- RotationSpeed = 2, rotationshastigheten
- Damage = 10, hur mycket skada missilen orsakar

Missilens _physics_process metod uppdaterar positionen för missilen med dess hastighet multiplicerat med delta. Sedan läser den in musens lokala koordinater och sparar den i variabeln mousePos. Sedan kör metoden en if kontroll som kollar om muspekarens absoluta Y-koordinater ör större än 30 och muspekarens X-koordinater är större än 10. Denna kontroll kollar om muspekaren är framför missilen men inte rakt framför. Alltså muspekaren måste vara endera 30 pixel till babord eller till styrbord om missilen och den måste vara 10 pixel framför missilen.

```
func _physics_process(delta):
	position += transform.x * speed * delta
	var mousePos = get_local_mouse_position()
	if abs(mousePos.y) > 30 && mousePos.x > 10:
		rotation += sign(mousePos.y) * rotationSpeed * delta
```

Figur 15fysik metoden för missilen

Missil scenen har också en _on_Misile_body_entered metod. Den här metoden är kopplad till Area2D nodens _body_entered signal. Det betyder att metoden kör varje gång som Area2D stöter på någon annan nod som den kan kollidera med. Som argument tar metoden den nod missilen har kolliderat med. Den fungerar igen med en if kontroll. Om noden den har kolliderat med har en metod som heter takeDamage så kallar på den metoden med Damage konstanten som argument. Slutligen så använder den queue_free() metoden för att ta bort missilen från spelplanen.

```
func _on_Missile_body_entered(body):
	if body.has_method("takeDamage"):
		body.takeDamage(damage)
	queue_free()
```

Figur 16kod för att registrera tröffar

4.3 Partikeleffekter

Nu har spelet alla fysiska objekt på spelplanen men spelet har inga visuella effekter. För att förverkliga det används partikeleffekter. Partikeleffekter är ett system i spelfysik,

rörelsegrafik och datorgrafik som använder många små till exempel sprite för att simulera vissa sorters suddiga fenomen, som explosioner, rök eller gnistor. Dessa fenomen är vanligen väldigt svåra att simulera med traditionella rendererings metoder.

4.3.1 Rymdskeppets motor

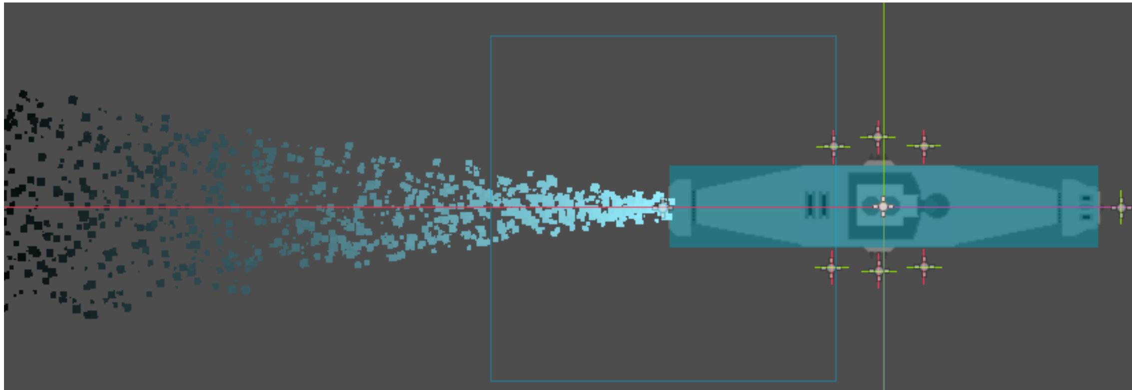
Rymdskeppets motorer behöver en partikeleffekt. Den förverkligas med en particles2D nod. Particles2D noden är en nod som producerar partiklar i en viss takt. Denna nod placeras där var motorn fins. Sedan måste den ha vissa specifikationer. Partikeleffekten för rymdskeppets motor använder följande egenskaper för Particles2D noden.

- Amount: 1000. Mängden partiklar som skapas per utsläpps cykel. Längen på en utsläpps cykel är samma som hur länge partiklarna lever. Så till exempel om partiklarnas livstid är en sekund är också utsläpps cykeln en sekund.
- Lifetime: 4. Fyra sekunders livstid. Varje partikel lever i fyra sekunder. Eftersom utsläpps cykeln är knuten till livstiden så skapas 1000 partiklar på 4 sekunder eller 250 partiklar per sekund.
- Local Coords: false. Detta betyder att partiklarna inte följer rymdskeppets koordinatsystem. Utan använder det globala koordinatsystemet. Orsaken för det är att partikeleffekten skall bli kvar på positonen där skeppet har varit och inte rakt bakom skeppet.

4.3.1.1 Process material

Partikeleffekten själv behöver ett ”material” som den använder. Det betyder att det specificeras mer hur partiklarna ser ut och beter sig. Som specifikationer användes följande.

- Emission shape: sphere, radius 7,21. Det betyder att partiklarna kommer ut i en cirkel med 7,21 radius.
- Spread: 9.68. hur noga partiklarna flyger i samma riktning.
- Gravity: 0. Att inte de skall falla åt något håll utan bara forsätta i samma riktning de börjat i.
- Initial velocity: 100, Farten på partiklarna när de genereras. Justeras av spelet på basen av rymbskeppet hastighet.
- Scale: 5, 1. 5 är storleken på partikeln och 1 hur mycket storleken varierar.
- Color ramp: Gradient. Specificerar 2 färger från ljusblå till svart. Desto längre partikeln har funnits desto svartare blir den.



Figur 17partikeleffekt

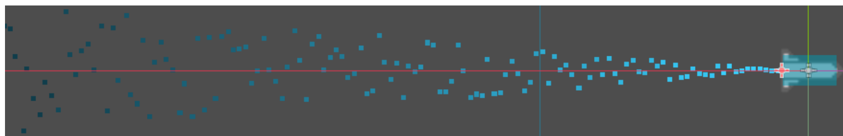
4.3.2 Missilens motor

Sedan skall missilens motor också ha en liknande partikeleffekt. Den förverkligas med samma Patricles2D nod men kräver andra egenskaper.

- Amount: 200.
- Lifetime: 1.
- Localcoords: False

4.3.2.1 Process Material

- Emission Shape: point. Alla partiklar kommer från en punkt.
- Spread: 5.
- Gravity: 0.
- Initial Velocity: 500. Den här farten är samma som missilens hastighet.
- Scale: 2, 0.
- Color ramp: Specificeras likadant som på rymdskeppets motor



Figur 18partikeleffekt flr missilen

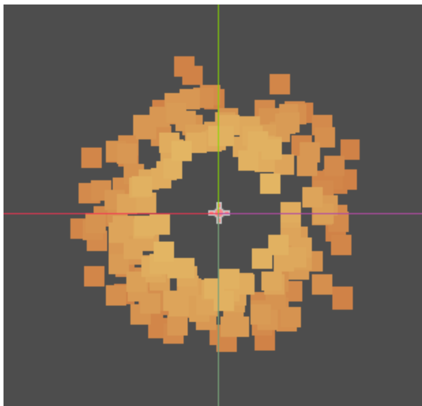
4.3.3 Explosion

När en missil träffar skall det också finnas en partikeleffekt. Den är byggd som sin egen scen. Denna scen aktiveras alltid när missilen träffar ett rymdskepp och är bara en Particles2D nod med följande egenskaper.

- Amount: 200
- Lifetime: 3
- Oneshot: true. Vilket betyder att den sker bara en gång och inte upprepar sig.
- Explosiveness: 0,8. Skalan är från 0,0 till 1,0. Om explosivness är 0,0 så skapas partiklarna i jämn takt i hela livstiden. Det passar bra för till exempel en motoreffekt. Om explosiveness är 1,0 så skapas alla partuklar genast i början av livstiden. Explosiveness 0,8 betyder att partiklarna skapas under den första 20 % av livstiden. Det ger en bra explosions effekt som inte sker exakt på samma gång.
- Localcoords: true.

4.3.3.1 Process Material

- Emissionshape: point.
- Spread: 180. Partiklarna flygar alltså i riktingen 180 grader till -180 grader. Det betyder att det blir en cirkel av explosionen
- Gravity: 0.
- Initial Velocity: 30
- Velocity rand: 0,3. hur slumpmässigt farten på partiklarna är.
- Scale: 5.
- Color ramp: Gradient. Men explosionens färger går från gul till mörkröd.



Figur 19partikelffekt för explosionen

4.4 Meny

I det här skedet är spelet byggt så att genast när man kör spelet så har du redan spelpjäsen på spelplanen och kan börja flyga omkring. För att det skall finnas funktionalitet för att starta spelet ensamt eller spela det med någon annan måste det finnas en meny där spelaren väljer hur spelet skall köras.
Den förverkligas genom att bygga en ny scen som heter Menu. Scenen själv är en **MarginContainer** nod. Denna nod ger en marginal åt alla dess subnoder. Det här betyder att knappar och dylikt som är subnoder av denna inte sitter fast i kanten av skärmen. Den här noden har en direkt subnod som heter **VBoxContainer**. Denna nod ordnar alla dess subnoder i en kolumn. Själva menyn ordnas som subnoder till den här noden. Den är uppbyggd av 2 olika sorters subnoder. Den första är en **Button** nod. Den är en standardnod för knappar som kan innehålla text eller ikoner. Den andra är en **LineEdit** nod. Denna nod ger funktionalitet för att skriva in en rad med text.
Noderna i meny är följande.

- PlayButton. En button nod. När man trycker på den startar den en ensam instans av spelet där du är den enda spelpjäsen på spelplanen.
- ServerButton. En button nod. När man trycker på den kör denna instans av spelet en server som andra spelare kan komma med i.
- JoinButton. En button nod. När man trycker på den tar denna instans av speletet kontakt med IP adressen som är inskriven i LineEdit noden.
- IpEdit. En LineEdit nod. Här skriver spelaren in IP adressen av spelet som kör servern.
- StarGameButton. En button nod. Den här knappen skall spelaren som kör servern trycka på när alla andra spelare har tagit kontakt med hans spel för att starta spelet.

4.4.3 Kod för menyn

Som först definieras här ett par konstanter.

- Const SERVER_PORT = 2024. Definierar vilken port som används när du skall kontakta andra instanser av spelet.

- Const MAX_PLAYERS = 2. Mängden av spelare som kan ta kontakt med en instans av spelet. I detta fall definieras 2 spelare för att testa funktionaliteten men kan ökas till 10.

Dessa konstanter har mer att göra med funktionaliteten för mångspelet men är definierade här.

4.4.3.1 PlayButton

Funktionaliteten för Play knappen är förverkligad med metoden _on_play_pressed(). I den här metoden definieras variabeln my_player. Denna variabel är en instans av rymdskeppscenen. Sedan lägger den till my_player som den enda spelpjäsen som skall skapas när spelet börjar till den globala playerToSpawn listan. Sedan använder den globala startGame() funktionen för att köra spelet och byta scen. Denna metod är sedan knuten till PlayButton nodens pressed() signal så att när man trycker på Play knappen so exekveras metoden.

```
func _on_play_pressed():
	var my_player = preload("res://Ship.tscn").instance()
	Global.playersToSpawn = [my_player]
	Global.startGame()
```

Figur 20kod för play knappen

4.4.3.2 ServerButton

Funktionaliteten för Start Server knappen är förverkligad med metoden _on_ServerButton_pressed(). I den här metoden definieras variabeln peer. Denna variabel använder NetworkedMultiplayerENet.new() metoden för att skapa en ny NetworkedMultiplayerENet instans. En NetworkedMultiplayerENet är en implementering av PacketPeer (Godot Docs - 3.3 branch, 2021). PacketPeer är ett protokoll för att kommunicera över UDP. Sen konfigureras det här nätverket som en server med SERVER_PORT och MAX_PLAYERS konstanterna som argument. Sedan sätter den peer variabeln som network_peer för spelet.

Denna metod är igen knuten till ServerButton nodens pressed() signal som exekveras när man trycker på knappen.

```
func _on_ServerButton_pressed():
	var peer = NetworkedMultiplayerENet.new()
	peer.create_server(SERVER_PORT, MAX_PLAYERS)
	get_tree().network_peer = peer
```

Figur 21kod för star a server knappen

4.4.3.3 JoinButton

Funktionaliteten för Join a Game knappen är förverklaigad med metoden _on_JoinButton_pressed(). I den här metoden definieras variabeln ip. Denna variabel tar innehållet VBoxContainer nodens LineEdit nod som argument och sparar det som IP. Sedan definierar den samma peer variabel som i ServerButton knappen där den skapar en ny instans av NetworkedMultiplayerENet. I det här fallet konfigurerar den det sedan som en klient. Med IP variabeln och SERVER_PORT konstanten som argument. Sedan sätter den igen peer variabeln som network_peer för spelet. Denna metod är igen knuten till JoinButton nodens pressed() signal som exekveras när man trycker på knappen.

```
func _on_JoinButton_pressed():
	var ip = $VBoxContainer/IpEdit.text
	var peer = NetworkedMultiplayerENet.new()
	peer.create_client(ip, SERVER_PORT)
	get_tree().network_peer = peer
```

Figur 22kod för Join a game knappen

4.4.3.4 StartGameButton

Funktionaliteten för Start Game kanppen är förverkligad med _on_StartGameButton_pressed() metoden. Den här metoden kör bra den globala startMultiplayerGame() metoden. Knuten igen till SatrGamebutton nodens pressed() signal.

```
func _on_StartGameButton_pressed():
	Global.startMultiplayerGame()
```

Figur 23kod för start a Game knappen

4.5 Multiplayer

Idén med det här spelet är att spelaren skall spela det mot andra spelare och inte mot spelet själv. Godot har ett High-Level API för multiplayer. High-Level API betyder att man inte behöver beblanda sig med detaljer för hur paket skickas över nätverket eller hur man skall

serialisera objekt. För att förverkliga det här måste spelet ha ett globalt skript där all funktionalitet programmeras. Alltså då skapas Global.gd skriptet

4.5.1 Global.dg

Global.gd skriptet är konfigurerat att köra som en klass som det bara finns en instans utav med Godots Autoload funktion. Det betyder att Godot skapar instansen av skriptet när du kör spelet. Orsaken för skriptet är att ha ett ställe som tar hand om logiken för Multiplayer. I själva scriptet definieras tre variabler.

- Player_info = {}. En tom Dictionary. Här sparas information om alla spelarna i Multiplayer spelet.
- My_info = { name = ”Johnson Magenta” }. En Dictionary med information om spelaren.
- playerToSpawn = []. En tom lista. I enna lista sparas alla rymdskepp för alla spelare som skall lägga still spelplanen när spelet börjar.

Först definieras metoden startGame(). Denna metod byter scenen från meny scenen till spelplanen.

```
func startGame():
	get_tree().change_scene("res://gameScene.tscn")
```

Figur 24metoden för att starta spelet

Nu definieras metoden _ready(). Denna metod kör när instansen av skriptet skapas. Den konfigurera sig själv att lyssna på 5 signaler. Signalerna kör sedan metoder enligt följande konfigurering:

- network_peer_connected kör metoden _player_connected
- network_peer_disconnected kör metoden _player_disconnected
- connected_to_server kör metoden _connected_ok
- connect_to_server kör metoden _connected_ok
- connection_failed kör metoden _connected_fail
- server_disconnected kör metoden _server_disconnected

```
func _ready():
	get_tree().connect("network_peer_connected", self, "_player_connected")
	get_tree().connect("network_peer_disconnected", self, "_player_disconnected")
	get_tree().connect("connected_to_server", self, "_connected_ok")
	get_tree().connect("connection_failed", self, "_connected_fail")
	get_tree().connect("server_disconnected", self, "_server_disconnected")
```

Figur 25kod för att lyssna på på nätverket

Dessa metoder definieras till näst. _player_connected metoden använder rpc_id metoden för att köra metoden register_player metoden på spelarens instans av spelet. Rpc_id metoden tillåter spelet köra kod i en annan instans av spelet som specificeras av spelarens id. _player_disconnected metoden tar som argument en spelares id och använder den för att ta bort informationen om spelaren från player_info. _conneccted_ok metoden printar ut ett log meddelande om att spelaren har lyckats kontakta servern. _server_disconnected printar ut et log meddelande om att kontakten till servern brutits och _coonnected_fail metoden printar ut ett log meddelande om att kontakten inte lyckades.

```
func _player_connected(id):
	rpc_id(id, "register_player", my_info)

func _player_disconnected(id):
	player_info.erase(id)

func _connected_ok():
	print("connected to server")

func _server_disconnected():
	print("server kick")

func _connected_fail():
	print("connection failed")
```

Figur 26metoderna som kallas från ready metoden

Nu definieras metoden register_player. Den är definierad som en remotesync metod vilket betyder att metoden kan köras via rpc och när man gör det så kör den också lokalt. Som argument tar den information om spelaren som skall registreras. Den använder metoden

get_rpc_sender_id() för att läsa av idn för den spelare som körde metoden och sparar den i variabeln id. Sedan sparar den spelarens information i player_info med idn som nyckel.

```
remotesync func register_player(info):
	var id = get_tree().get_rpc_sender_id()
	player_info[id] = info
```

Figur 27metoden att registrera spelare

Nu definieras en till remotesync metod pre_configure_game. Det första den gör är att använda set_pause metoden för att pausa spelet till alla är färdiga. Sedan använder den metoden get_network_unique_id() för att få spelarens id och sparar den i variabeln selfPeerID. Sedan sätter den playerToSpawn variabeln till en tom lista. Nu skapar den en instans av rymdskepp scenen. Som namn för scenen använder den selfPeerID. Detta gör den för att varje rymdskepp skall ha ett unikt namn. Sedan sätter den selPeerID som network_master för rymdskeppet. Spelaren med den idn är spelaren som kontrollerar rymdskeppet. Sedan lägger den till skeppet till listan playerToSpawn. Sedan loopar den över alla spelare i player_info. För varje spelare så skapar den instans av rymdskeppet, sätter spelarens id som namnet på skeppet, sätter spelaren som network master för skeppet, och lägger skeppet till playersToSpawn listan. Till slut så ändrar den scenen till gameScene och kör metoden done_preconfiguring för spelaren med id 1. Spelaren som är servern har alltid id 1.

```
remotesync func pre_configure_game():
	get_tree().set_pause(true)
	var selfPeerID = get_tree().get_network_unique_id()

	Global.playersToSpawn = []

	var my_player = preload("res://Ship.tscn").instance()
	my_player.set_name(str(selfPeerID))
	my_player.set_network_master(selfPeerID)
	Global.playersToSpawn.append(my_player)

	for p in player_info:
		var player = preload("res://Ship.tscn").instance()
		player.set_name(str(p))
		player.set_network_master(p)
		Global.playersToSpawn.append(player)
		
	get_tree().change_scene("res://gameScene.tscn")
	rpc_id(1, "done_preconfiguring")
```

Figur 28metoden för att konfigurera spelet

Nu definieras remotesync metoden done_preconfiguring(). Först så läser den idn för den spelare som kallade metoden med metoden get_rpc_sender_id() ocha sparar den i variabeln who. Sedan kollar den att koden kör på servern med metoden is_network_server() och att spelaren som kallade metoden done_preconfiguring inte redan finns i listan på spelaren som är färdiga. Sedan lägger den till spelaren till listan på färdiga spelare. Om listan på klara spelare är lika stor som listan på alla spelare så kör den metoden post_configure_game för alla spelare.

```
remotesync func done_preconfiguring():
	var who = get_tree().get_rpc_sender_id()
	assert(get_tree().is_network_server())
	#assert(who in player_info)
	assert(not who in players_done)

	players_done.append(who)

	if players_done.size() == player_info.size():
		rpc("post_configure_game")
```

Figur 29metoden för när spelet har konfigurerat

Nu definieras remotesync metoden done_preconfiguring. Den kollar om rpc idn är lika med 1, alltså att det är servern som kör koden. Om det är servern så använder den set_pause metoden för att stänga av pausen på spelet. Om det inte är servern så gör den ingenting.

```
remotesync func post_configure_game():
	if 1 == get_tree().get_rpc_sender_id():
		get_tree().set_pause(false)
```

Figur 30metod för att stänga av pausen på spälet

Sista metoden som definieras är startMultiplayerGame. Den kollar om det är servern som kör metoden. Om inte så gör den ingenting. Om det är servern så säger den åt alla att köra metoden pre_configure_game.

```
func startMultiplayerGame():
	if not get_tree().is_network_server():
		return
			
	rpc("pre_configure_game")
```

Figur 31metod för att starta spelet

5 RESULTAT

5.1 Vad som har uppnåtts

Det här spelet har funktionaliteten att starta spelet var efter spelaren ser en meny. Om spelaren trycker på Play knappen så får det spelet att byta scenen till spelplanen med ett skepp som spelaren kan styra. Här kan spelaren skjuta missiler och styra dem med muspekaren. Missilerna följer muspekaren enda till de har passerat muspekaren varefter de flyger rakt. Om spelaren med hjälp av muspekaren styr missilerna tillbaka in i sitt eget rymdskepp så exploderar missilerna och rymdskeppet tar skada och du ser en grafisk visualisering på hur mycket skada rymdskeppet har tagit. Om rymdskeppet tar tillräckligt med skada så byter spelet scenen tillbaka till meny scenen.
Från meny scenen kan också spelaren starta en server. Därefter kan en annan instans av spelet gå in på server som kör spelet genom att skriva in IP adressen och trycka på Join a Game knappen. När den andra spelaren är på plats så kan instansen av spelet som kör servern kan trycka på knappen Start Game, vilket startar spelet och får båda instanserna av spelet att byta till spelplanscenen.

5.2 Vad som inte har uppnåtts

Multiplayer modulen fungerar inte som den borde. När en spelare spelar multiplayer med två spelare så skapas två skepp i spelarens egen instans. I spelarens egen instans av spelet reagerar båda rymdskeppen till indata som spelaren ger. Det vill säga att om båda skeppen följer kommandon av spelaren, men varken eller av rymdskeppen i den andra instansen följer spelarens kommandon. Detta sker också för den andra instansen av spelet och instanserna är inte synkroniserade. Spelet alltså kastar båda spelarna in i sin egen spelplan där de styr 2 rymdskepp och om ena dör har det ingen effekt på den andra spelarens spel. Om denna funktionalitet skulle fungera så skulle spelet ännu inte ha något definierade lag d.v.s. att det skulle vara alla spelare mot alla spelare. En spekulerad orsak varför multiplayer modulen inte fungerar är för att alla skepp blir registrerade som samma id för att id är baserad på spelarens IP adress och när du kör 2 instanser av spelet med smma IP

address så registrerar den båda skeppen till samma instans. Detta problem kan möjligen fixas med att köra instanserna av spelet på 2 olika datorer.

6 SLUTSATSER

Syfte med detta examensarbete var att kolla hur godot spelmotoorn fungerar och att bygga ett enkelt multiplayer-spel med motorn. Spelet jag byggt här bör vara ett enkelt spel att bygga i vilken som helst spelmotor. Det borde till och med gå att bygga i enkla spelbyggarpogram som GameMaker(”yoyogames.com” 2021). Resultaten som jag nådde var slutligen ganska enkla att åstadkomma även om Multiplayer modulen jag byggde inte fungerar som planerat. En av orsakerna som Multiplayer modulen inte fungerar upptäckte jag att enligt Global.gd skriptet jag skrev så får varje spelare samma IP adress vilket orsakar till varför båda spelpjäserna styrs av en spelare. Vad jag inte har upptäckt är varför spelet jag byggde kastar båda spelarna in i sin egen spelplan som inte synkroniserar med varandra.

Godot är en spelmotor som lämpar sig bra för att laga 2D spel. Dess integrering av hela processen in i en och samma miljö göra det enkelt för grupper av utvecklare att hålla koll på sitt projekt. Den är för tillfället ännu mera siktat mot nybörjarspelutvecklare. Hoppeligen med mera tid kan vara den blir en bra konkurrent till Unity och Unreal.

Har jag skapat en bra guide för att någon annan skall kunna laga samma spel som jag med hjälp av detta arbete? Jo och nej. För att producera ett spel med funktionaliteten där du kan flyga omkring i ett rymdskepp och skjuta missiler så ger detta arbete god vägledning. Men det som ännu är obesvarat är multiplayer modulen. Själva förklarning på hur den förverkligas är mycket stökig dels p.g.a. att bygga den var mycket stökigt vis. Godot själv skulle behöva mer dokumentation på hur en multiplayer modul borde byggas. Jag är ännu inte övertygad om att alla metoder är definierade på rätta platser och jag vet att koden för att namnge spelarna går i kårs.

7 KÄLLFÖRTECKNING

Patreon godot engine, 2021. Tillgänglig: https://www.patreon.com/godotengine Hämtad: 10.4.2021

Godot Docs - 3.3 branch, 2021. Tillgänglig: https://docs.godotengine.org/en/stable/index.html Hämtad: 12.5.2021

Godot Engine - Features, 2021. Tillgänglig: https://godotengine.org/features Hämtad: 12.5.2021

Kids can code - Godot Recipes, 2021 Tillgänglig: http://kidscancode.org/godot_recipes/ Hämtad: 12.5.2021

Godot 3 Tutorial Series, 2021, Gamefromscratch. Tillgänglig: https://www.youtube.com/watch?v=iDEcP8Mc-7s&list=PLS9MbmO_ssyDk79j9ewONxV88fD5e_o5d&index=2 Hämtad 5.3.2021

Piskelapp, 2021 Tillgänglig: https://www.piskelapp.com/ Hämtad: 10.1.2021

GameMaker, 2021 Tillgänglig: https://www.yoyogames.com Hämtad: 18.5.2021

Weber ,D. 1994, *On Basilisk Station,* Baen Books

Peckham, E., 2019, How Unity built the world's most popular game engine, TechCrunch 17.10.2019

Thomsen, M., 2012, History of the Unreal Engine, IGN 15.6.2012

Linietsky, J,. 2014, Godot Engine reaches 1.0, first stable release, godotengine.org 15.12.2014

Linietsky, J,. 2014, Godot history in images 1.0, first stable release, godotengine.org 15.1.2014

Etcheverry I.R., 2017, Introducing C# in Godot, godotengine.org 21.10.2017

Berman, D,. 2020 What is the MIT License? Top 10 questions answered, snyk.io 26.7.2020