Zombie Survivor

Spel- & kontroll-beskrivning (inkl. eventuella spelinstruktioner)

Spelet är ett förstapersons actionspel där spelarens uppdrag är att rädda gisslan från attackerande zombier och eskortera dem till en säker räddningszon. Under spelets gång genereras ett obegränsat antal zombier som aktivt försöker döda både spelaren och gisslan. För att vinna krävs det att spelaren lyckas rädda ett visst antal gisslan, medan spelet förloras om spelaren själv eller för många av gisslan dör. Spelaren är utrustad med ett vapen med begränsad ammunition samt en kniv för nödsituationer. Det finns powerups för att återställa spelarens hälsa spridda runt banan.

Designmönster för många zombies:

- Data Oriented Technology Stack (DOTS) för extrema mängder objekt
- Flyweight (Scriptable objects)

Tabell av kriterier

Kommunicerad Spelidé och Målstruktur

Spelets grundidé kommuniceras tydligt till spelaren redan från början genom en introscen som beskriver att målet är att rädda gisslan från zombier och eskortera dem till en säker plats. Det narrativa gör det enkelt för spelaren att förstå både varför och hur spelet ska spelas. Under spelets gång förstärks detta mål visuellt och funktionellt via UI-element som visar spelarinformation. Denna strukturella tydlighet ger spelaren kontinuerlig feedback på progression och vad som återstår för att vinna eller riskera att förlora. Genom att kombinera berättelse, UI och spelmekanik säkerställs att målet alltid är närvarande i spelarens medvetande, vilket skapar ett tydligt ramverk för beslut och handling.

Design för Engagemang och "Flow

För att skapa engagemang och ett naturligt speltempo har vi byggt spelet med kontinuerligt inkommande zombier, vilket tvingar spelaren att anpassa sin spelstil. Spelaren uppmuntras att röra sig aktivt mellan räddningspunkter och hotfulla zoner, vilket förhindrar passivt spelande. Feedbacksystemet visuellt (exempelvis animationer vid skada mot zombie), ljudmässigt (vapenljud) samt UI-baserat (hälsa, ammunition) förstärker handlingarna och ökar känslan av kontroll och inlevelse. Spelkontrollerna är medvetet simpla och responsiva, vilket gör att spelaren snabbt kan nå ett tillstånd av "flow" där fokus ligger på strategi och reaktion, snarare än att hantera gränssnitt eller teknik.

Spelarens Agens och Meningsfull Interaktion

Spelaren har stor kontroll över hur spelet utvecklas genom meningsfulla val som påverkar både strategi och resultat. Det finns flera sätt att närma sig spelets mål: till exempel kan spelaren välja att prioritera att rädda gisslan snabbt, eller stanna och bekämpa zombier. Valet mellan att använda pistol eller kniv beroende på tillgången till ammunition innebär att spelaren aktivt måste anpassa sig till sin situation. Möjligheten att ignorera eller samla powerups introducerar också taktiska beslut. Denna typ av spelarfrihet skapar en mer personlig upplevelse och förstärker spelarens upplevelse av att vara aktivt delaktig i berättelsen och dess utfall.

Fördjupad Tillämpning/Experiment

Vi har experimenterat med flera system som går utanför grundläggande mekanik, för att öka djupet i spelet. Vi har infört ett rudimentärt “dismemberment” system där spelaren kan skjuta av delar av zombierna för att mer visuellt visa skada som man gör. Tanken var att kroppsdelarna skulle ramla av helt som sina egna objekt, men detta stretchade zombiens modell så den idén skrotades.

A. Speldesign & Teoretisk Tillämpning - 20p max, 10p min. Samlade poäng: 14p			
Poäng	Kriterium	Beskrivning	Kommentar
4	Kommunicerad Spelidé och Målstruktur	Spelets grundidé, tema, mål och eventuellt slutskede kommuniceras tydligt för spelaren (t.ex. via UI, miljö, narrativ). Strukturen stödjer spelarens förståelse och framåtrörelse.	Introscen med bakgrundsinfo för spelaren, sätter upp en berättelse med tydligt mål och motivering.
4	Design för Engagemang och "Flow"	Medvetna designval (t.ex. utmaningskurva, feedbacksystem, spelartempo, kontroller) som syftar till att skapa och bibehålla spelarens engagemang och känsla av "flow".	Rädda gisslan i en utmanande miljö.
4	Spelarens Agens och Meningsfull Interaktion	Spelaren ges möjligheter att göra val eller utföra handlingar som upplevs meningsfulla och har märkbar påverkan på spelet eller spelupplevelsen.	Skjuta, ladda om, röra sig fritt, byta vapen, rädda gisslan.
2-8	Fördjupad Tillämpning/Experiment	Implementering och motivering av en eller flera mer avancerade speldesignmekaniker, tydlig tillämpning av specifika spelteorier, eller experiment med narrativa/interaktiva system (t.ex. dynamisk svårighet, komplex ekonomi/progression, konsekvensrika val, unika narrativa	Zombie dismemberment.

		grepp). Poäng baseras på komplexitet, integration och dokumenterad koppling till teori/principer. Godkännande av kursansvarig kan behövas för ovanliga förslag.	

B. Spelmekanik & Användarupplevelse - 30p max, 15p min. Samlade poäng: 21p			
Poäng	Kriterium	Beskrivning	Kommentar
3	Spelbar huvud-loop	Spelbar från start till tydligt vinst- och/eller förlust-tillstånd utan blockerande buggar; kärnmekaniken ger omedelbar spelarfeedback (visuell, auditiv).	Huvudmeny med mappsystem och intro till en bana. Vinst eller förlust där man kan börja om.
3	Nivådesign (om tillämpligt)	Minst en nivå designad med tydlig struktur, start, mål och medvetet placerade utmaningar/element.	1 nivå med olika områden i olika höjder.
2	Samlarobjekt / Belöningar	Minst en typ av samlarobjekt eller belöningssystem med tydlig effekt/syfte (t.ex. poäng, hälsa, nyckel, resurs).	Health. HP Powerup.
3	Spelarkaraktär – 3D-kontroll	Välfungerande 3D Player Controller med grundläggande rörelse (t.ex. hopp, coyote-time, slope-limit, step-offset om relevant).	FPS-Player med hopp och checks för ground och water. Även mekanik för att kunna skjuta och göra skada på fiender. HP med 3 hjärtan.
1	Grundläggande UI-feedback	Minst ett dynamiskt UI-element (t.ex. hälsa, poäng, timer) som uppdateras under spel.	Health Ammo
2	Ljud & Musik	Relevanta SFX och bakgrundsmusik, implementerade via Audio Mixer-grupper.	Bakgrundsljud och effekter på eldstad och pistol.
3	Visuell Presentation & Polish	Partikelsystem, post-processing och/eller konsekvent art-style som höjer helhetsintrycket. Visuell spelarfeedback vid viktiga händelser.	Partikeleffekter: Blod när en zombie blir skjuten. Eldstad.
3	3D-kamera med Occlusion-hantering	Cinemachine-kamera som hanterar occlusion (undviker clipping), ev. med dolly-zoom i trånga utrymmen, för god spelupplevelse.	

2	3D-spatialt ljud	3D spatialt ljud med panorering, roll-off och doppler för djupkänsla.	Eldstad spatialSound.
2	Tillgänglighetsaspekt	Minst en medveten design/funktion för ökad tillgänglighet (t.ex. FOV-reglage, kameraskak av/på, textstorlek, färgblindfilter)	Hjärtan blinkar vid skada.
1-5	Valfri UX/Mekanik-Feature	Egen UX-/spelmekanik-funktion (t.ex. unika spelarförmågor, in-game instruktioner/tutorial, enklare dialogsystem, prestationssystem, publicering online via t.ex. itch.io). Poäng & godkännande bestäms av kursansvarig (maila)	

C. Teknisk implementation - 30p max, 15p min. Samlade poäng: 20			
Poäng	Kriterium	Beskrivning	Kommentar
2	Rigidbody-fysik	Rigidbody-baserade fysikinteraktioner (t.ex. knuffa lådor, ramla, explosioner).	Zombie ragdoll, player
2	Colliders	Minst tre olika collider-typer (t.ex. för mark, väggar, objekt, fiender) ändamålsenligt använda.	WaterCollider, och övriga colliders för ground samt fiender och gisslan.
1	Prefabs	Prefabs som återanvänds effektivt via kod eller i scen (t.ex. projektiler, fiender, pickups).	Zombies, gisslan
3	Input System	Unitys Input System (Input Actions) för tangentbord, mus & gamepad.	Playermovement
1	Cinemachine (utöver grundläggande i B)	Avancerad Cinemachine-användning (t.ex. state driven cameras, smarta övergångar, specifika effekter).	
2	Animationer	Minst 3 animation clips via Animator Controller, som bidrar till spelet.	Zombies använder sig av animationer med hjälp av state machine med triggers och avatar mask för att smälta samman animationer (rörelse + attack/ta skada samtidigt) Pistol rör sig när man skjuter och

			laddar om.
3	Sparfunktion	Sparfunktion för t.ex. high-score, position, upplåsta nivåer (via PlayerPrefs/JSON)	
3	NavMesh-AI pathfinding	NavMesh AI pathfinding för fiender/NPCs i komplex 3D-geometri (inkl. NavMesh Links om relevant).	NPC:s Nav Mesh Agent
3	Shader Graph / Custom Shader	Egen shader via Shader Graph eller kod (t.ex. fresnel, dissolve, toon, visuella effekter).	
3	Belysning & Skuggor (Baked + Real-time)	Medveten användning av baked (lightmapping, prober) och/eller real-time belysning/skuggor för stämning/prestanda.	Vattenspegling,
2	Object Pooling	Object pooling för återanvändning av ofta spawnade objekt (minskar GC-spikes).	
2	Event-system	Event-system (C# events/UnityEvents) för löskoppling mellan system (t.ex. Gameplay ↔ UI).	Introscen
2	Scenhantering	Asynkron scenladdning (LoadSceneAsync) med t.ex. loading-screen/snygga övergångar.	
3	Procedurellt innehåll	Procedurell generering av meningsfullt innehåll (t.ex. nivå, loot); poäng baserat på komplexitet och integration.	
3	ECS / Data-orienterad design (Teori/Enkel tillämpning)	Förståelse för och eventuell enkel tillämpning av principer för DOTS/ECS eller liknande dataorienterade mönster.	
1-3	VR eller AR Spel	Skapa ett spel i Virtual Reality eller Augmented Reality. Oculus Quest 3 finns att låna (kontakta kursansvarig vid intresse).	
1-5	Egen Teknisk Feature	Egen teknisk feature/innovation. Poäng & godkännande via kursansvarig.	Enkel dismemberment system

Reflektion över arbetsprocess & lärdomar

Under projektets gång har vi som grupp fått rätt bra praktiskt erfarenhet av Unity då vi undvek de problem som dök upp i första labben. Vi började projektet med att fördela arbetsområden baserat på intresse som hjälpte oss att arbeta mer fokuserat och effektivt. Det blev tydligt att alla roller hängde ihop på ett eller annat sätt, då fick vi prata samman om vad som behöver göras mellan våra kodstycken till exempel.
Våran uppdelning såg ut såhär:

Ronnie - Bana/ljud
Christian T - vapen/hostage
Simon - zombies/navmesh
Nasera - powerups/viewmodel/UI
Chrille V - spelaren

Våra problemlösningsstrategier bestod mestadels av parprogrammering eller att dela skärm för att felsöka tillsammans. Det var ett arbetssätt som fungerade bra och sparade mycket tid, särskilt när någon fastnade i ett tekniskt problem. Vi märkte att det var mycket lättare att snabbt lösa buggar tillsammans än att sitta enskilt och försöka gissa sig fram.

Det var också nyttigt att bygga ett system där olika komponenter kunde kommunicera med varandra, t.ex. när en zombie gör en attack och något ska ta skada (spelaren eller gisslan). Där lärde vi oss vikten av att tänka "modulärt", så att system inte blir beroende av varandra på ett krångligt sätt. Istället försökte vi skapa logiska gränssnitt och använda managers som kunde hantera övergripande logik. Detta gjorde det enklare att återanvända kod och bygga vidare på funktioner utan att behöva skriva om allt från grunden.

Överlag har vi utvecklat en bättre förståelse för Unity som helhetsplattform, från animation och fysik till UI och ljud. En annan viktig lärdom var vikten av att hålla koden modulär och flexibel. Ju längre in i projektet vi kom, desto mer insåg vi hur mycket lättare det blev att bygga vidare på något som från början hade skrivits med framtiden i åtanke.

Reflektion över Git-arbetsflöde

När det gäller Git och versionshantering så märkte vi förra labben att det var en utmaning att hålla allt organiserat, särskilt i början och slutet där vi märkte att scene-filer var den största boven för problem. I denna labben separerade vi oss totalt i våra egna mappar och scener inuti projektet vilket ledde dels till att det var enklare att se vad var och en hade jobbat med men också att hålla logiken fysiskt separerad. Alla hade varsin scen att testa sitt arbete på utan konsekvens för de andra.

För att hjälpa oss komma igång och ha en överblick använde vi ett enkelt kanban-board med issues, där vi delade upp arbetsområden och kunde bocka av vad som gjorts. Det hjälpte särskilt i början när det var mycket nytt att sätta upp. Även om vi inte följde en strikt Scrum-process eller hade dagliga standups, så fungerade det här upplägget bra för att

snabbt komma in i ett flyt. Alla kunde se vad som var på gång och vi kunde lättare synka om någon fastnade eller behövde hjälp.

Vi använde oss endast av pull requests istället för att pusha direkt till main, just för att undvika konflikter och ha bättre koll på vad som faktiskt blev implementerat. Vi utförde i princip väldigt enkla code reviews fast oftast mot våra egna pull requests.