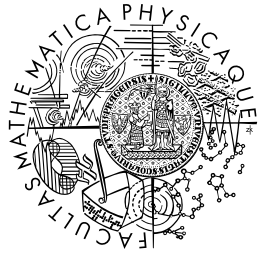


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Oleg Petruny

Vícežánrová příběhová počítačová hra s podporou načítání nového obsahu za běhu

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Martin Mirbauer
Studijní program: Informatika (B0613A140006)
Studijní obor: IPP6 (0613RA1400060010)

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rozhodně děkuji svému vedoucímu Mgr. Martinu Mirbauerovi za jeho čas, ochotu, trpělivost a cenné rady. Velký dík patří také mnoha dalším lidem uvedeným v titulcích hry za jejich pomoc, testování a podporu. Bez všech těchto lidí by tato práce buď vůbec nevznikla, nebo by nedosáhla takového rozsahu a kvality.

Název práce: Vícežánrová příběhová počítačová hra s podporou načítání nového obsahu za běhu

Autor: Oleg Petruny

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Martin Mirbauer, Katedra softwaru a výuky informatiky

Abstrakt: V dějinách herního průmyslu je opravdu malé množství her, schopných načítat obsah za běhu, natož když hry jsou s příběhem a určené pro jednoho hráče. Tato práce poskytuje vzorovou implementaci obsahlé hry v Uneral Engine, 3D modelů a dynamického soundtracku. Výstupem je dohratelný celistvý příběh probíhající v pět žánrově a dynamicky odlišných úrovních. Projekt zavádí rozhraní pro načítání nového obsahu za běhu, systémy dialogů, cutscén, quick time eventů a objektové interakce. Spolu s tím jsou podrobně řešeny problémy a úskalí při tvorbě hry a grafiky v Unreal Engine 5.

Klíčová slova: Počítačová hra Unreal Engine dynamické načítání obsahu 3D grafika

Title: Multi-genre game with support for loading new content in real-time

Author: Oleg Petruny

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Martin Mirbauer, Department of Software and Computer Science Education

Abstract: In the history of the gaming industry, there are really few games capable of loading content on the fly, let alone when the games are story-driven and designed for a single player. This thesis provides an example implementation of a comprehensive game in Unreal Engine, 3D models, and a dynamic soundtrack. The output is a playable, coherent story pieced into five genre and dynamic different levels. The project introduces an interface for loading new content on the fly, systems of dialogues, cutscenes, quick time events, and object interaction. Along with this, in detail are addressed problems and pitfalls in creating a game and graphics in Unreal Engine 5.

Keywords: Computer game Unreal Engine dynamic content loading 3D graphics

Obsah

Úvod

Běžně příběhové hry jsou navrhnuté předem a zůstávají neměnné. Existují ale i výjimky, například hry žánru rogue-like, kde ale stále generování obsahu je založeno na použití speciálně vytvořeného a odladěného algoritmu. Tento algoritmus různé složitosti stále používá pouze prostředky předem navržené a odladěné vývojáři. To vše má podstatný důvod, jelikož vývojáři mají za úkol vytvořit tzv. dobrý herní zážitek a zaručit, že hra bude umělecky správně interpretovaná hráčem. Bohužel, nezávisle na kvalitě výsledné hry – počet opakovaného zahrání hry v nejlepším případě klesá nebo v tom horším je pouze jeden.

Celý problém v branži příběhových her je řešen tzv. mody neboli modifikacemi hry a to různých druhů, jako například grafické, kódové a další. Samotní vývojáři poskytují potřebné prostředky pro podporu nebo tvorbu modů. Dokonce i technicky zdatní hráči dokážou oblíbené hry úpravit, aby ony jakousi podporu měly. V každém případě modifikace značně zvětšují počet přehrání her, kde klasickým příkladem je hra The Elder Scrolls: Skyrim od studia Bethesda.

Přestože celé řešení zdánlivě funguje, spoléhá se na ochotu samotných hráčů módy vytvářet. Jestliže hra nebude mít dostatek hráčů, potom taktéž nebude mít dostatek nového fanouškovského obsahu a výsledně počet přehrání neroste. Nedavným příkladem je hra Starfield od již zmíněného studia Bethesda. Navíc k mnoha technickým omezením samotných módů, musí mít hráči nejen ochotu, ale i být technicky a umělecky zdatní, aby něco vytvořili. Minimálně musí vynaložit úsilí k manuálnímu vyhledání, stažení a instalaci modů.

Tato práce se zaměřuje na tvorbu hry a návrh systému který umožní výše popsaný lidský faktor a nedostatky eliminovat. Přináší tak příběh rozdělený na pět žánrově odlišných úrovní a zavádí high-level API pro Unreal Engine na stahování a načítání obsahu do hry přímo za běhu. Specifický příběh snižuje ludonarativní disonanci[1] při vzniku nového obsahu ve hře a přítomnost více žánrů umožňuje otestovat, jestli toto řešení v každém z nich dostatečně funguje. Zároveň tato práce přenechává samotné generování obsahu pomoci AI modelů a testování výsledků až jako další rozšíření. Čili primárně se zaměřuje na kostru samotné hry, aby spotom byl prostor, kam nový obsah začlenit.

[1] Souvislost resp. logické propojení herního světa, příběhu a gameplaye.

Hlavními tématy, na které se práce zaměřuje, jsou:

- Práce s Unreal Engine, jeho reálná omezení, obcházení/vyrovnání se s těmito omezení a tipy.
- Postupy tvorby různých druhů grafiky pro 3D hry zejména v UE.
- Postupy tvorby zvuků a hudby pro hry.
- Ukázkové příklady tvorby herních systémů a mechanik pro Unreal Engine.
- Tvorba generativního obsahu a jeho načítaní na Unreal Engine.

Kapitola 1

Zásady tvorby počítačových her

Vývoj her je obor, který je často mylně interpretován širokou veřejností. Existuje všeobecná představa o tom, jak by hry měly vypadat, jaké prvky musí či nesmí obsahovat, kdo a jak je má vyvíjet a jaká by měla být jejich cenová politika. Ve skutečnosti však průměrného uživatele zajímá především míra *zábavy* a herní zážitek, který za investovaný čas a finanční prostředky získá.

Z pohledu hráče nejsou faktory jako pracovní podmínky vývojářů nebo kontroverzní herní mechaniky primárními rozhodovacími kritérii. Klíčovým aspektem je optimalizace uživatelského zážitku a zajištění dostatečné interaktivity, plynulosti a vtažení do děje, které přímo ovlivňují retenci a herní ekonomiku.

Jelikož je zábava vysoce subjektivním konceptem, neexistuje univerzální model hry, který by vyhovoval všem uživatelům. Herní design proto využívá analytických metod, jako jsou uživatelské testování, behaviorální analýza a iterativní vývoj, aby maximalizoval pozitivní odezvu v cílové skupině hráčů.

1.1 Průběh vývoje

Pro nikoho snad není překvapivé, že pro vznik díla je potřeba nějaká myšlenka, která mu předchází. V tomto jsou hry stejný jako jiné kreativní odvětví. Je potřeba, aby myšlenka byla funkční, a v našem případě zábavná. Bohužel hry už s většinou odvětví nesdílí aspekt, při kterém funkčnost myšlenky lze ověřit již na začátku produkce. A je to ještě horší, protože to můžeme většinou zjistit pouze u konce. Nezanedbatelná část her proto potom je zamítnutá, neviditelná nebo dokonce předělána hned před koncem. O jednom takovém případu – který skončil šťastně – doporučuji si přečíst v knize *Blood, Sweat, and Pixels* [1] o hře Uncharted 4. Autor převypráví rozhovory s vývojáři her různých žánrů a velikostí, které znázorňují unikátní a zároveň společné překážky v oboru vývoje videoher.

1.1.1 Design dokument

Nedílnou součástí vývoje hry je iterace nápadů a celkový popis prostředí a způsobů produkce. To vše v sobě zahrnuje Design dokument. Ten může mít různé podoby, jelikož jeho hlavním cílem je uchovat potřebné nápady na snadno dostupném a přehledném místě. Souhrn těchto nápadů, jejích propracovanost a to, jak dobře spolu fungují, potom tvoří celou hru i výslednou zábavu. Proto dobré písemné zpracování pomůže předat myšlenky designéra a jejich aktuální verzi celému týmu vývojářů.

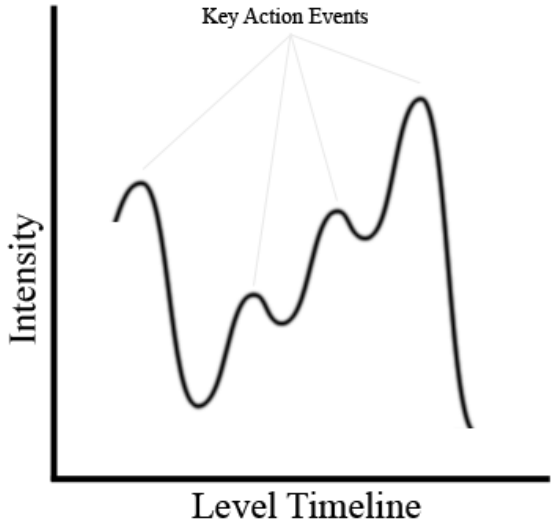
Design dokument, jenž tvoří nedílnou součást této práce, byl vytvořen s ohledem na vývoj hry jedním autorem. Z tohoto důvodu jsou v něm jednotlivé nápady popsány poměrně stručně až abstraktně, neboť sloužil primárně jako osobní vodítko. V případě týmového vývoje by však bylo vhodné dokument rozšířit o podrobnější popis nápadů a jejich zamýšlené realizace.

1.2 Téma, motivy, příběh, cíl

Hra by měla poskytovat nějakou *myšlenku* relevantní pro hráče, aby ten měl zájem si ji zahrát. Počáteční myšlenku většinou vytváří a následně rozvíjí game designer. Ta následovně může být zachována i na konci vývoje, nebo být zásadně změněná samotnou hrou, jejími mechanikami a kreativním provedením. Krásný případ je známá desková hra „Landlord's Game" od Lizzie Magie později známá jako „Monopoly". Paní Magiová chtěla vytvořit hru, která by hráčům představila hrůzy kapitalismu a monopolů tak, že by poskytla hráčům zkušenosti, kde jeden jedinec neustále a nevyhnutelně bohatne zatímco zbytek stejně tak neustále a nevyhnutelně chudne. Bohužel hráči moc tuhle myšlenku nepochitili. Místo vystrašení z kapitalismu, naopak rádi zažívali hazard a možnost ekonomicky zruinovat soupeře.

Způsobů, jak předat myšlenku hráči, je nespočetně mnoho. Pro účely této práce proto probereme možnosti předáni myšlenky v příběhových hrách. Ty by měly mít nějaké ústřední téma, které se následně obohacuje příběhy okolo. Je velmi důležité, aby okolní příběhy měly *motivy*, aby následně i hráč měl motiv je prožít. Dokonce hráč nemusí znát motivy až do konce hry, stačí aby byla cítit smysluplná návaznost a pocit možné odměny. Proto je za umění považováno i velkolepé předstírání existence motivů, které donutí hráče *strávit ve hře co největší množství času.*

Samotná tvorba příběhů je poněkud podobná tvorbě knižní nebo kinematografické. Jen je zapotřebí oživovat a propojovat nejen postavy a svět, ale i herní mechaniky s ohledem na jejich zábavnost, složitost a technická omezení.



Obrázek 1.1 Ukázkový graf představující distribuci klíčových momentů a cesty k ním. Převzato z článku *Gameplay Fundamentals Revisited: Harnessed Pacing & Intensity* [2].

Údržba pozornosti Cíle a jejich distribuce pomáhají hře udržet si hráče. Pokud například za sebou následuje několik "nudných" pasáží, nebo je hra příliš repetativní, hráč může ztratit zájem. Obrázek 1.1 znázorňuje ukázkový příklad pěkné distribuce klíčových momentů. Co přesně jsou klíčové momenty, je na vývojáři. Mohou to být důležité momenty v příběhu, vylepšení postavy hráče, představení nové důležité mechaniky atd. Důležité je nechat hráče v každém okamžiku pocítit jeho úspěchy na začátku tohoto momentu nebo jeho konci. Je to dost abstraktní, ale přesto fungující způsob jak udržet pozornost hráče a poskytnout mu zábavu.

1.3 Žánr, mechaniky, reference, platforma

Důležité je také rozmyslet si vhodné umělecké a technické aspekty hry. Například nebude moc zábavné hrát hlavní mechaniku farmářství, pokud hlavní žánr hry je horor. Správná volba žánru a mechanik je tedy klíčová pro vytvoření konzistentního a poutavého herního zážitku.

Žánr Žánr hry určuje její základní atmosféru, pravidla a často i cílovou skupinu hráčů. Většinou se dnes setkáváme s hybridními žánry, které dokážou poskytnout hráči více obsahu a tím pádem i více možné zábavy.

Při výběru žánru je důležité vzít v úvahu preferované herní mechaniky a jejich složitost. Například hra zaměřená na rychlou a dynamickou akci bude pravděpodobně obsahovat prvky střílečky nebo bojové hry, zatímco narativně založená hra může využívat prvky adventury či RPG.

Herní mechaniky Mechaniky jsou základní interaktivní prvky, které hráč využívá k postupu ve hře. Spravný návrh mechanik zajistí, že hra bude plynulá, intuitivní a zábavná.

Při jejich návrhu je třeba zvážit: jak mechaniky podporují zvolený žánr, jak se budou vyvíjet v průběhu hry a jak se kombinují s ostatními prvky hry. Například v hororové hře bude dobře fungovat správa omezených zdrojů pro zvětšení napětí, zatímco ve strategické hře budou klíčové rozhodovací prvky a řízení ekonomiky.

Platforma Platforma ovlivňuje technické aspekty vývoje i celkový dosah hry mezi hráči. Každá platforma má své specifické a občas i náročné požadavky. Za to ale odmění hráče vhodnějším ovládáním nebo unikátním vizuálním zážitkem.

Při výběru platformy je nutné zvážit technická omezení a očekávání hráčů na dané platformě. Například mobilní hry často využívají dotykové ovládání a krátké herní smyčky, zatímco hry pro PC a konzole mohou nabídnout komplexnější mechaniky a delší herní dobu.

Kopírování Kopírování cizích a vlastních nápadů je nedílnou součástí úspěšného vývoje. Hodně se vyplatí mít přehled ve vybraném žánru a mechanikách. Není nic špatného učit se na chybách a úspěších jiných her. Důležité ale je si pamatovat, že neexistuje deterministický vzorec, jak vytvořit dokonalou kombinaci příběhů, žánrů a mechanik tak, aby hra získala oblibu hráčů.

Minihry Skvělou metodou, jak rozptýlit hráče od monotonie herního cyklu jsou minihry. Přitom je lze aplikovat kdykoliv. Minihry většinou buď poskytují odměnu, anebo slouží k relaxaci mezi náročnějšími segmenty hry. Důležité je, aby jejich design byl v souladu s celkovým stylem hry a nepůsobil rušivě. Například rytmická minihra ve fantasy RPG může být zajímavým doplňkem, ale v realistické hororové hře by působila nepatřičně.

1.4 Engine

Jakmile je rozhodnuto, jak bude hra vypadat, je zapotřebí zvolit vhodné prostředí pro její vývoj. Vývojář sice může vytvořit vlastní herní engine, avšak v dnešní době to zpravidla znamená jen zbytečné komplikace. Proto jsou na trhu dnes již běžně dostupná hotová řešení, která pokrývají většinu potřeb.

Základní funkce herních enginů obvykle zahrnují editor scény, přehrávače a čtečky různých multimediálních či digitálních formátů, simulaci fyziky, zpracování uživatelského vstupu a zajištění kompatibility výstupu mezi různými operačními systémy a hardwarem. Nad rámec toho většina enginů poskytuje i vlastní univerzální implementaci herních objektů a mechanik. Například primitivní chůze herní postavy, či implementace renderingu obrazu a grafických prvků jako osvětlení, stíny, odrazy a případně další. Tedy většinou se jedná o technologie, které se využívají poměrně často, ale jejich implementace bývá časově náročná až nevýhodná.

Vytvářet hru pomoci enginu může mít i své nevýhody. Některé technologie, zejména grafické, mohou být vynucené, čili pevně svázané s daným enginem, čímž vynucují určité způsoby použití a tím pádem omezují kreativní svobodu. Kromě toho většina enginů není distribuována zcela zdarma. Vývojáři buď musí předem uhradit licenční poplatky za middleware, nebo jsou vázáni na procentuální odvody z výdělků, což je dnes častější model.

Tato práce je zamýšlena jako vývoj 3D příběhové hry s více žánry a s možností dynamického načítání nového obsahu za běhu. Pro tyto účely se nabízí čtyři hlavní enginy na trhu, přičemž výběr v předchozí práci padl na Unreal Engine a proto táto navazujiící práce v tom pokračuje. Z tohoto důvodu veškerý následující sekce, věnované technologickému rozboru či implementaci, se zaměří výhradně na možnosti, které Unreal Engine nabízí.

Unity[1] Unity je nejspíš stále nejpopulárnější volbou v roce vzniku této práce. Lze na něm vytvořit hru libovolného žánru a rozsahu. Má rozhodně největší komunitu a rozsáhle návody, jednodušší programovací jazyk C# a skriptování herních objektů. Zároveň již obsahuje možnost načítání obsahu za běhu, zmíněnou v požadavcích.

Má ale své problémy, které pozorný vývojář procití už v polovině vývoje jestli ne dřív. Tvorba dobré grafiky často vyžaduje psaní vlastních HLSL shaderů, což moc nekoreluje s jednoduchostí C#. Navíc grafika často vyžaduje psaní vlastních optimalizačních algoritmů nebo manuální adaptaci cizích pluginů. Následovně i samotný C# vytváří komplikace s rychlostí běhu programu, obsahem zabrané paměti a přítomností garbage collectoru. Všechny tyto problémy lze čás-

[1] https://unity.com/

tečně opravit a vylepšit, jen je potřeba mnoho pokročilých znalostí navíc. Detaily a zkušenosti vývojářů lze nalézt online, například zde[3].

V roce 2019, kdy vznikalá předchozí práce, v Unity byl také problém s paralelizací hlavního cyklu samotného enginu a rendrovacích úloh. Byl používán triviální model herní smyčky, který spouštěl logiku sekvenčně v hlavním vlákně. Stejně tak bylo triviální i spouštění renderovacích úloh bez dynamického clusteringu a paralelizace. V celku Unity tehdy ještě nebyl tak robustní a nenabízel tolik vývojářských možností resp. oprav. V den vydání této navazující práce, je pravděpododně nejlepší volbou pro jednotlivce ale i malé týmy. Proto pro tuto práci bychom dnes zvolili Unity, kdybychom už neměli předchozí prototyp postavěný na Unreal Engine.

Unity je engine otevřený všem žánrům a mnoha platformam jako mobilní, VR a další. Často je používán v menších projektech a menších týmech například při vývoji indie her, ale taky i při vývoji velkých her.

Godot[2] Godot je velmi diskutovaná novinka, která se celkem dobře šíří trhem. Hlavní výhoda je jeho drobnost a že je úplně zdarma za všech okolností. Godot je totiž open source produkt vyvíjený komunitou a dokonce jeho vývoj je podpořen některými herními studii či jinými firmy.

Má podobné problémy jako jiné enginy a něco navíc nefunguje dobře nebo chybí. Například výchozí implementace fyziky a kolizí není často dostačující nebo nepředvídatelně a uživatelsky špatně řeší určité okamžiky. Je ale vskutku ohromující, co vše může nabídnout. I když většinu nabízených technologií vývojář potřebuje přepsat vlastní rukou, jelikož často nefungují, jak je zapotřebí. Příklady jsou k nahlédnutí online na forumech[4] nebo taky blozích[5].

V době vzniku hry přiložené k práci byl Godot ještě příliš nový a nevypadal nijak perspektivně. V den vydání této práce, je solidní konkurent ve 2D herních žánrech jako platformery, logické hry či mobilní hry.

Unreal Engine[3] Unreal Engine verze 4 byl kdysi zvolen enginem pro hru, z které potom vznikla tato práce. Jeho primární výhodou oproti konkurenci jsou vysoce standardizované postupy neboli pipelines, které zlevňují nebo vůbec umožňují tvorby her ve velkých týmech. Celý engine se navíc chlubí velkými „úspěchy" v různých technologiích, zejména rendrovacích – což probereme v další sekci – a taky odvětvích jako motion design, kinematografie, design architektury a další.

Nabízené možností jsou vskutku ohromující, když se snažíte vybrat budoucí stavební kámen pro vaši hru. Je ale potřeba brát v úvahu silnou nepřívětivost

[2] https://godotengine.org/

[3] https://www.unrealengine.com/

enginu k nováčkům, kteří chtějí udělat něco vlastního mimo již existující návody. Stejné lze říct o oficiální dokumentaci, která je často a velmi nedostačující – jestli vůbec existuje. Navíc při snaze udělat něco kompletního pomocí nabízených pokročilých technologií, v závěru vývojář musí dané technologie ovládnout na velmi vysoké úrovni a často i modifikovat zdrojový kód enginu. V některých případech technologie jako Nanite nebo Lumen nejde použit pro dokonalé a odladěné výsledky, proto se prostě zahazují – probereme v další sekci.

Pokud ignorujeme chybějící dokumentaci, je Unreal Engine stejný engine jako ostatní. Některé technologie má nesrovnatelně lepší a některé horší. Lze najít návody a neoficiální dokumentaci diky velké komunitě. Dokonce vývojář může samotný engine upravovat podle sebe, protože lze zcela zdarma dostat přístup ke zdrojovým souborům. Velké týmy tuto nezávislost s radostí využívají.

Aktuálně UE se orientuje na 3D hry převážně s grafikou vysoké kvality a stejně jako Unity podporuje většinu aktuálních platforem. Taky se skvěle hodí pro tvorbu filmu, motion design a realtime simulace. Navíc díky technologiím jako Nanite a Lumen začína přebírat trh architektonických rendereru.

CryEngine[4] CryEngine je hodně podobný Unreal Enginu za výjimkou toho, že se specialuzuje jen na vývoj her. Taky již není tolik univerzalní, dokumentace je ještě míň, komunita je velmi malá a přívětivost je snad nejhorší možná. Je to dost úzce specializovaný engine, který potřebuje silné odborníky k jeho ovládání.

Všechny známé hry na CE jsou zaměřené na velmi propracovanou grafiku a často hry s mechanikami boje nebo střelby od první osoby. Nedokázal jsem dohledat žádné 2D hry nebo indie, což je pochopitelné. 2D hry není specializace CryEngine a tvořit indie na tomto enginu je neefektivní až nepřínosné. Taky často vývojáři her na CE se přiznávají, že je nutné přizpůsobovat zdrojový kód enginu podle vlastních potřeb, což ještě víc odrazuje začátečníky.

1.4.1 Programovácí jazyk

Unreal Engine umožňuje programovat pomoci vizuálních bloků tzv. Blueprintů nebo textově v jazyce C++. I při použití pouze jedné z variant lze vyvinout celou hru. Veškerý potenciál se ale projevuje při jejich kombinaci. V C++ se skvěle programují komplexní a nízkoúrovňové prvky, zatímco Blueprinty jsou skvělé pro skriptování úrovní a objektů pomocí high-level bloků. Samozřejmě samotné vizuální bloky lze v C++ vlastnoručně vytvořit.

Je to vše zároveň velmi obsáhlý aspekt Unreal Engine. Začátečníkovi bude celkem dlouho trvat než začne sám něco vymýšlet. Na první pohled jednoduché

[4] https://www.cryengine.com/

vizuální programování je ve skutečností velmi komplexní už samotnou téměř nekonečnou nabídkou funkcí.

Blueprinty Blueprinty jsou skvělé pro rychlé a jednoduché skriptování. Umožňuje to do procesu programování hry zapojit členy týmu z jiných méně technických odvětví. Potom například level designer může iterovat vlastní nápady mnohem rychleji a nečekat na programátora, tím že si poskládá vlastní logiku a hned ji může vyzkoušet. Spolu s jednoduchostí jsou zároveň velmi bezpečné. Nefunkční kód se za běhu jen poznamená do logů a engine nebude havarovat.

Samozřejmě to přináší nějaký overhead, hlavně když se jedná o práci s velkými daty. Pokud ale udržovat Blueprinty s malým počtem funkcí a používat reference místo kopírování dat, potom je overhead zanedbatelný.

Největší nevýhodou je překvapivě nepřehlednost kódu, která nejčastěji značí špatný programátorský návrh nebo lenost autora. Totiž velký počet vizuálních bloků a jejich propojení není možné umístit přehledně na jednu obrazovku. To potom vyúsťuje v nepřehledný mix různých logických částí, které jsou dost obtížné na orientaci a údržbu. Napravit to nejde ani rozdělením jedné funkce do více funkcí v samotném Blueprintu. Nepřehledný mix propojení bloků se pouze změní na nepřehledný mix oken s ruzným kódem. Správný postup v tomto případě je ručně konvertovat logiku z Blueprintu do C++.

C++ Práce s C++ v Unreal Engine je hodně podobná práci s velkými frameworky jako například Qt. Použití čistého C++ je zcela povolené, ale takový kód potom nelze použít v blueprintech a tedy i editoru. Unreal Engine proto definuje speciální makra jako například UCLASS a UFUNCTION pro možnost integrování kódu buď přímo blueprintu nebo aspoň systému reflexe. Makra se potom zpracovávají ne macro preprocessor, ale Unreal Header Tool nebo Unreal Build Tool, které slouží jako generatory kódu. Generatory potom sami generují potřebné funkce a proměnné pro systém reflexe a editor.

V C++ a navíc s otevřeným kódem celého enginu, má vývojář plnou kontrolu nad během programu nebo jeho debugováním. Problém je ale použití assetů z editoru nebo reference objektů v herním světě. Jsou možnosti jak to obejít, například statické načtení assetu z registru pomoci konstantní plné cesty k assetu nebo přeiterovat všechny objekty ve světě. Editor samořejmě není schopen takové reference udržovat v případě přemístění assetu a časté iterování přes všechny objekty je citelná zátěž. Proto ve většíně případů je potřeba zpřístupnít celou třídu do Blueprintu a v editoru rovněž vytvořit Blueprint podtřídu, která bude pouze přiřazovat potřebné reference.

1.4.2 Grafika

V rukou máme taktéž širokou nabídku technologií a nástrojů pro tvorbu grafiky nebo import do hry grafického obsahu vytvořeného v jiném softwaru. Je ale potřeba dát si záležet na veškerých nastavení enginu. Těch nastavení je velké množství a rozhodně se budou iterovat během celého vývoje hry. Výchozí nastavení totiž jsou příliš univerzalní a nekompletní. Přestože i v základu s nimi hra vypadá na dostatečné úrovní, při hraní bude stálý pocit nedokončenosti produktu nebo kopírování cizího díla. Většina vývojářů si totiž nedá zaležet, jak jejich hra vypadá kvůli úspoře času nebo technické náročnosti tohoto kroku. Proto se mnoho her (postavených na Unreal Engine) vizuálně a "pocitově" podobá, což je mnoha hráčům nepříjemné.

- Materiály

Materiál je odborné označení souboru dat a funkcí, které reprezentují výsledný vzhled objektu, efektu, světla nebo post-processu vyrenderovaného snímku. Je to de-facto vysokourovňový shader, který, i přestože je dost omezený, dokáže generovat skvělý a kreativní výstup. Případně lze zdrojový kód materiálu poskytnutého enginem rozšířit o potřebné funkcionality a v případě potřeby napsat i vlastní render pipeline, jak to často dělají nekterá studia.

Substrate[5] Substrate je nově vyvíjený systém materiálů, který se primárně chlubí jednoduchostí vrstvení materiálů a celkového návrhu. Vrstvení textur a materiálů je jistě pohodlné a podobá se klasickému vrstvení v grafických editorech. Umožňuje to tvorbu komplexních a nádherných povrchů za relativně málo úsilí.

Problémem však je zjednodušenost systému se zaměřením na pohodlí generického grafického umělce. Vytváří to ještě víc omezení pro grafické a technické možností a téměř úplnou nepoužitelnost technických triků.

HLSL shadery V Unreal Engine lze napsat a aplikovat HLSL shadery. Je to ale malo dokumentovaný aspekt a zahrnuje spoustu práce okolo, narozdil od toho jak lehce to lze zprovoznit v Unity.

[5] https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-substrate-materials-in-unreal-engine

- Osvětlení

Unreal Engine vyniká bohatým výběrem možností provedení osvětlení. Lze zde najít různé druhy přímých a nepřímých zdrojů světel, technik odrazů a stínů. Nebo taktéž velké množství objemných prostorů pro ovlivňování světla jako mlhy nebo rozptyl a paprsky. Všechno lze relativně podrobně nastavit buď pro účely výkonu nebo grafické estetiky.

Lumen[6] Lumen je zdánlivě revoluce v realtime herním osvětlení. Jedná se o speciální režim osvětlení a odrazů, který umožňuje za běhu počítat dynamické osvětlení bez potřeby vytváření lightmap nebo pracného nastavení ploch odrazů. Vytváří velice kvalitní osvětlení při vynaložení minimálního úsili.

Nevýhodu, kterou ale již autoři UE neprezentují, není jen výrazně větší zátěž na hardware, ale i šum vyrendrovaného osvětlení a odrazů. Lumen totíž v reálném čase generuje buffer s odrazy a osvětlením ve velice malém rozlišení a navíc vynechává náhodně zhruba polovinu pixelů. Potom se výsledný buffer rescaluje na potřebné rozlišení a aplikuje. Celý tento systém se spoléhá na zhlazovací metodu Temporal Anti-Aliasing, která akumuluje výsledky předchozích snímků a interpoluje je do aktuálního. Tak potom v klidných scénach šum je zdanlivě dobře potlačen i když ne kompletně. Ale v dynamických scénach neboli rychlejším pohybu kamery, TAA vytváří ghosting efekt nejen na hranach objektů ale i osvetlení s odrazy, který často hráčům je nepřijemný.

- 3D

Podporován je import 3D modelů z různých formátů. Navíc k tomu UE poskytuje dostatečné množství optimalizačních technik jako např.:

- Render occlusion culling vykreslující objekty pouze pokud se nachází v zorném poli kamery, nebo
- Level Of Details (LOD) vykreslující různě detailizované verze objektu v závislosti na jeho vzdalenosti od hráče, protože není potřeba vykreslovat detaily, které z dálky nejsou vidět.

[6] https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine

Nanite[7] Nanite je další specialita Unreal Engine 5, která umožňuje vykreslování a instancování objektů s miliony až miliardami trojúhelníků v reálném čase. Umožňuje tak použití v enginu extremně detailních objektů, získaných pomoci skenování objektů v realitě tzv. fotogrammetrie.

Technika je založená na clustarizaci objektů při importu a následně dynamickém streamování pouze viditelných trojúhelníků. Bohužel neumožňuje kompletní vynechání LOD systému, jak to inzeruje tvůrce enginu. Nanite sice může ušetřit spoustu času modeláři, ale reálný zisk ve výkonu je pouze ve velmi náročných scénach obsahující statické objekty. Proto LOD stále zůstává nejefektivnější technikou zkrácení vykreslovacího času objektu.

Instancování Instancování umožňuje zmenšení paměti potřebné pro reprezentaci skupiny stejných objektů a výsledně i zkrácení renderovacího času. Je postavená na jednoduché myšlence, že instance objektů drží v paměti pouze transformaci, a zbytek dat je referencován z jedíneho pravého objektu. Často se tato technika používá pro naplnění světa různými drobnými objekty (instancování) nebo tvorbu vegetace (foliáž).

- Animace

Veškeré potřeby pro animaci libovolného druhu jsou taktéž pokryté.

Skeleton Skeleton animace je založená doslovně na animování kostry přivázané k 3D modelu. Skupiny trojúhelníků jsou namapované na určité kosti, tak že při pohybu kosti je stejným směrem interpolovaná pozice trojúhelníků. Daný druh 3D animace může vytvářet velkou zátěž na procesor zejména při velkém počtu instancí, jelikož ten musí propočítávat pohyb každé kostí vůči hernímu světu a až potom odesílat renderovací požadavek na grafiku. Přesto daná technika tvoří převážnou část všech animací ve hrách díky jednoduchosti tvorby a práci s ní.

Shader Shader animace je exotická technika s myšlenkou přeskočit krok s výpočtem na procesoru. Animace se zakóduje do textury a následně je aplikovaná ve vertex shaderu jako offset vrcholů. Výsledně lze například velmi levně animovat velké hejno ptáků, které nemusí mít kolizi ve světě.

[7] https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine

- Renderer

Z rendererů lze vybrat mezi Forward a Deferred rendererem. Jsou to systémy, které se starají o zpracování grafiky a vykreslování scény. Zajišťují převod 3D objektů a jejich vlastností na obraz, který se zobrazí na monitoru. Renderer pracuje s osvětlením, stíny a materiály pro dosažení realistické nebo stylizované grafiky. V Unreal Engine ovlivňuje také dostupné prostředky pro post-processing.

Forward rendering Forward rendering je metoda renderingu, kde objekty se renderují pro každý zdroj světla zvlášť a výsledné osvětlení je součtem těchto průchodů. Je vhodný pro scény s malým množstvím světel a vyžaduje menší paměťovou náročnost. Dnes se převážně používá v mobilních a VR aplikacích, kde je důležitá nízka latence. Nevýhodou je například nemožnost použití Screen Space vykreslovacích technik a omezení na maximalně 4 dynamických světel na objekt – 4 kanály resp. RGBA.

Deferred rendering Deferred rendering odkládá aplikaci světelných efektů a v první fázi ukládá informace o geometrii scény do bufferu (G-buffer). Světla se aplikují až v druhé fázi, což umožňuje efektivnější práci s větším množstvím dynamických světelných zdrojů. Tato metoda se používá především v moderních AAA hrách, kde je vyžadována vysoká vizuální kvalita. Je to výchozí renderer v Unreal Engine, který navíc umožňuje použití technologií Nanite a Lumen.

Nevýhodou je nemožnost použití kvalitních zhlazovacích metod jako Multisample Anti-Aliasing. MSAA funguje tak, že ukládá průměr více vzorků na pixel během rasterizace, což dobře funguje u forward renderingu, kde se barva a hloubka určují okamžitě. V deferred renderingu se však barvy a osvětlení aplikují až v pozdější fázi, kdy se světelné výpočty provádějí na pixelech na základě uložených dat v G-bufferu. Problém je, že MSAA by muselo být aplikováno na všechny jednotlivé buffery (normály, hloubku, albedo atd.), což je extrémně výpočetně náročné a neefektivní. Proto se místo MSAA často používají jiné metody zhlazování, jako FXAA, SMAA nebo TAA. Ty jsou podstatně méně kvalitní a TAA dokonce způsobuje rozmazávání hran objektů v dynamických scénach resp. ghosting efekt.

Zároveň takova metoda není schopná poskytnout některé grafické techniky. Například transparentní materiály (např. skla) nebo subsurface scattering (rozptyl světla při průchodu objektem nebo dopadu na něj) se musí renderovat na konci ještě jedním průchodem tzv. forward passem.

- Postprocessing

Postprocessing je technika dodatečného zpracování vyrenderovaného obrazu. Většinou se jedná o triviální manipulaci barev, ale lze tady dělat i spoustu dizajnových a technickych triků. Např. různé glitch efekty, efekty tepelného/nočního vidění atd. získané pomocí procedurální nebo statické modulace obrazu. V deffered rendereru navíc lze používat screen space techniky, které používají vyrenderovaný snímek k rychlé tvorbě odrazů (SSR - Screen Space Reflections) nebo zastínění (SSAO - Screen Space Ambient Occlusion).

- 2D

Je více způsobů práce s 2D grafikou, avšak udělat čistě 2D hru bude problematické. V oficiální nabídce je rozhraní Paper 2D[8], které vývoj 2D hry sice zjednodušuje, ale přesto má pouze základní prvky 2D enginu. Místy chybí optimalizace, protože soubory v editoru jsou neracionálně velké a ve větších projektech engine začíná být hodně náročný na hardware. Proto se pro 2D hry Unreal Engine moc nehodí a je doporučené rozhlédnout se po konkurenčních enginech.

UI UI lze programovat v C++ pomoci třídy Slate nebo přímo pomoci Blueprintů v editoru. Programování se Slate je hodně nízkoúrovňové tedy stejné jako slepé programování okenní windows aplikace pomocí Win32 API. Mnohem pohodlnější je práce v editoru, kde rovnou lze vidět výsledek. Celkově tvorba UI v Unreal Engine je jedna z jeho nejsilnějších stránek mezi konkurenty, i když se o ní moc nemluví.

Pro zobrazení UI nebo jiných 2D prvků ve 3D bylo vždy možné vytvořit klasickou plochu s texturou/materiálem nebo renderovat text do 3D světa. Samotné renderování textu ve 3D je implementováno pomoci generace 3D objektu z vektorového fontu, což je často výkonnostně přehnané řešení neboť mesh texty jsou již součástí hotového modelu. Ještě lze použít renderování textu z předgenerované průhledné rastrové textury fontu. Poslední způsob je dost rychlý na implementaci a výkonnostně nejlepší pokud výstup nemusí být nějak extra kvalitní. Nejlepší možností je generovat text vektorově v UI a následně ho promítnout do 3D světa, což je umožněno pomoci již zmíněných UI tříd.

[8] https://dev.epicgames.com/documentation/en-us/unreal-engine/paper-2d-overview-in-unreal-engine

1.4.3 Zvuk

Pro práci se zvuky máme taktéž bohaté možností. Důležité je předem nastavit kompresi a způsob streamování zvukových dat z assetů hry. Ještě lepší je rozdělit zvuky do menších spojitých balíčků assetů. Často totiž chceme zvukovou stopu použít hned v okamžik nějaké udalostí, a je důležité, aby se data co nejrychleji načetla. Samozřejmě taky lze některé zvuky načíst předem a držet v paměti procesu. Ale i když se nezdá, zvuky mohou obsadit poměrně velký kus paměti aplikace a většinou zbytečně, což také většina vývojářů ignoruje.

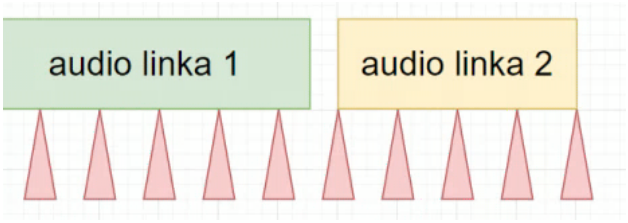
Mixer a Cue Cue je chytrým kontejnerem pro jeden a více konkrétních zvuků, který se chová jako jeden samostatný zvuk. V tomto kontejneru lze zvuky mixovat nebo větvit, modifikovat tonalitu, hlasitost, modulaci, dělat přechody a mnoho dalšího. Navíc veškeré efekty a jejich intenzitu lze aplikovat staticky nebo dynamicky. Dohromady vše prochází přes Mixer, který funguje jako zjednodušený mixážní pult. Nastavuje a kombinuje přiřazené zvukové třídy a může na nich aplikovat equalizer.

Dynamická hlasitost Attenuation je struktura parametrů popisující modifikaci zvuku při rozdílné vzdálenosti posluchače od zdroje zvuku. Dohromady tak popisuje, na jakou vzdálenost lze zvuk slyšet a jak bude znít ztlumení a zesílení. Parametry podrobně upřesňují, jak se zvuk bude šířit, blokovat, odrážet a splývat v prostoru. Většinou je postačující pouze první základní skupina parametrů popisující objem zvukového prostoru a funkci, která provádí interpolaci hlasitosti.

1.4.4 Hudba

Během existence Unreal Engine 4 neexistoval téměř žádný oficiální nástroj pro dynamickou ani statickou kompozici hudebních linek. Proto vzníkl jednočlenný tým, který daný nástroj programoval a dokonce byl dostupný v alfa verzi ke stažení. S příchodem Unreal Engine 5 práce na nástroji byla pozastavena a tedy žádný oficiální nástroj stále neexistuje a není v plánu.

Hlavní problém s hudbou je synchronizace linek. Hudba je dost náchylná na jakékoliv zpoždění neboli desynchronizaci linek, protože i milisekunda rozdílu může přeměnit nádherně zkomponovanou hudbu v neposlouchatelnou kaši. Proto jednoduché přehrání linky v potřebný okamžik nefunguje. Buď se zvukový data načtou pomalu nebo herní tik provede spuštění opožděně, protože je závislý na renderovácí frekvenci snímků viz. Obrázek 1.2 na další straně.



Obrázek 1.2 Diagram znázorňující spožděné spuštění navazující hudební linky z důvodu závislosti spouštění na snímkové frekvenci hry.

FMOD[9] FMOD je middleware audio engine pro hry poskytující velice silné rozhraní pro práci se zvuky nebo jejich manipulaci. Nejčastěji se k němu přiklání pro tvorbu dynamických hudebních doprovodů, což vyplňuje hudební mezeru v UE. V populárních enginech má integrovanou podporu nebo poskytuje engine v podobě samostatného procesu, který potom může komunikovat s procesem hry. FMOD také poskytuje editor zvukových stop a s nimi spojených eventů, pro jednoduché ovládání audia přímo ve hře.

1.4.5 Načítání obsahu

V Unreal Engine načítání obsahu je realizované v podobě herních patchů. Umožnuje to exportovat a nahrát libovolný asset nebo kód, což je potom znázorněno v praktické části práce. Nevýhodou takového přístupu je dlouha iterace a aplikace patchů v případě jejích většího množství – oficiální dokumentace doporučuje nepřesahovat mez v 100 patchů.

Zdlouhavé načítání Načtení i těch menších assetů může značně pozastavit hru. Proto je potřeba načítání nového obsahu provést hned při načtení celé hry, nebo zapracovat nad vhodným malým streamingem dat, který nezpůsobí velkou zátěž na hardware.

Kompatibilita Kompatibilita nového a starého kódu je něco, co může zhavarovat celou aplikaci. Naštěstí UE řeší havarijní stavy a hra v případě načtení nekompatibilního nebo poškozeného obsahu poběží dál, ale potřebné assety se již nenačtou. Nejvíc se tento problém projevuje s klasickým kódem v C++, kde výsledný strojový kód očekává nějakou proměnnou nebo funkci v přesně daném místě paměti. O něco lepší je to se vším ostatním, protože vše je propojeno a voláno relativními nebo globálními cestami a názvy. Tak například funkce v blueprintech jsou vždy voláné pomoci názvů funkcí v řetězcové podobě. Stejně tak proměnné

[9] https://www.fmod.com/

jsou uložené v runtime generované pomocné třídě. Takové využití víceúrovňové reflexe, umožňuje ošetřit libovolný problém s kompatibilitou a zaručit bezpečný běh za cenu trochu většího zatížení prostředků. Navíc stejnou reflexi v C++ lze jednoduše udělat pomoci maker, které Unreal Engine poskytuje. Ovšem ošetření kompatibility není součástí této práce.

1.4.6 Umělá inteligence herních postav

Nehratelné postavy resp. NPC jsou součástí většiny her, protože pomáhají vyprávět příběh nebo oživit prostředí. NPC jako každý herní prvek lze napevno navrhnout a naprogramovat, určit všechny potřebné varianty vzhledů a použití. Ale z designových a časových důvodů ve většině případu chceme, aby NPC byli více univerzální. Chceme aby se mohli samostatně orientovat v prostředí, pohybovat se a občas dokonce reagovat na okolí a ovlivňovat ho.

Behavior tree Strom pravidel neboli Behavior tree je nejčastěší způsob kódování chování NPC. Takový strom umožňuje efektivní rozhodování a řízení chování umělé inteligence. Hlavní výhodou oproti jiným přístupům, jako například konečné automaty nebo plánování, je modularita, škálovatelnost, snadná údržba a současně přehlednost. Samozřejmě má i nevýhody, mezi které zejména patří optimalizace a náročný návrh složitých víceúrovňových chování.

NPC začíná rozhodování v kořenu stromu, od kterého postupuje k vrcholům s pravidly. Listy jsou vždy akční vrcholy a, jak napovídá název, určují akci, kterou objekt provede. Cestu od kořene k listům vytváří řídicí vrcholy, každý z nich určuje pořadí a podmínky přechodu na podřízené vrcholy. Nejčastější jsou:

- sekvenční resp. Sequence, který cyklicky spouští své podřízené vrcholy postupně, dokud některý nevrátí přerušení,
- selektor resp. Selector, vybírá první podřízené vrchol, který v zadaném pořadí má pozitivně zhodnocené pravidlo,
- paralelní resp. Parallel, který spouští podřízený vrcholy současně,
- a podmínkové vrcholy, které rozhodují, zda pokračovat v dané větvi, nebo se vrátit ke kořenu.

Pomocí těchto pravidel lze popsat mnoho různých chování postav neboli objektů. UE navíc poskytuje užitečné systémy pro pokročilé chování "inteligentních" objektů, jako například systémy koordinace pohybu. Postava se tak může řídit navigační síti (navmesh), která napovídá objektům místo a způsob obcházení překážek. K tomu jsou k dispozici i pokročilejší implementace reakcí chytrých objektů na zvuký, obraz a jiné objekty.

1.5 Umělá inteligence pro tvorbu obsahu

Již dnes se objevují hry, které integrují danou technologii. NPC tak umí poměrně realisticky a nekonečně držet konverzaci s hráčem v textové a dokonce i hlasové podobě s definovaným typem osobností. Možnosti generování textur a spritů jsou na pohled snad nekonečné. Vývojáři mohou poměrně rychle vygenerovat jednoduchý kód nebo velké konfigurační soubory a struktury. Navíc již existuje a zdokonaluje se generování 3D objektů, videí a hudby. V teorii stejná umělá inteligence je schopná generovat popis a rozmistění obejktů v herním světě nebo vyprávět příběh.

V současné době díky výkonostním pokrokům se rozšiřují hranice použitelností umělé intelegence a to zejména velkých jazykových modelů (Large Language Models). LLM ukazují skvělé výsledky v generování obsahu různého charakteru v poměru lidského času a ceny.

Celé toto téma je technicky a odborně velice zkrácené s ohledem na rozsah bakalářské práce. Jednotlivé modely a práce s nimi nejsou součástí této práce. Tato práce zakládá pouze potřebný koncept pro integraci umělé inteligence.

Udržení kontextu Udržení kontextu je největší problém této technologie. Nejvíce se to projevuje při generování videí, kde je zřetelné jak model má potíže udržet konkrétní obsah nebo myšlenku jednotlivých scén a následně i vzhled vizuálních objektů. LLM s každou další iteraci má poměrně vysokou šanci změnit směr „myšlenky". A to proto, že LLM nemyslí, LLM je pouze násobení velkého množství matic pravděpodobnostní tzv. váhové transformery. Tak například LLM nevytváří logicky souvislý text ale pouze predikuje další pravděpodobnostně nejvhodnější sekvence textu na základě trénovacích dat.

Proto zásadní chybou je použití LLM k úplné tvorbě nového obsahu. A správné je používat LLM pouze jako nástroj buď pro tvorbu počáteční verze obsahu nebo vedlejší obohacení již existujícího díla. Právě s touto myšlenkou byl doprovázen vznik této práce, díky níž bude možné otestovat, jak dobře LLM bude obohacovat již hotovou počítačovou hru.

Halucinace Další významnou slabinou LLM jsou halucinace. Jedná se o situace, kdy model generuje nesprávné, zavádějící nebo zcela smyšlené informace. V kontextu počítačových her tak může docházet například k halucinacím při generování herních dialogů, příběhu nebo vizuálních prvků, kde model vymýšlí neexistující herní mechaniky, postavy či události, které nepatří do hry. Kořen tohoto problému je stejný jak v udržení kontextu, kde navíc se to prohlubuje slabým natrenováním modelu.

Řešení halucinací nejsou dokonalá, ale mohou výrazně zlepšit stav problémů. K dispozici je větší množství metod a některé dokonce proprietární. V teorii se ale většinou jedná o metody:

- retrieval-augmented generation (RAG), která kombinuje model s databází obsahující ověřené informace,
- jemné doladění modelu resp. fine-tuning, které spočívá v dotrenování modelu na specifických datech (na datech naši hry),
- a omezení generativní volnosti pomocí pravidel a pevně definovaných scénářů, které se zároveň kryjí s kontrolními mechanismy a pomocí validace výstupů.

1.6 Tato práce navazuje na předchozí

Původně drobné demo hry bylo již vytvořené za účelem obhajoby maturitní práce. Předchozí práce ale měla globálně jinou myšlenku a motivaci. Aktuálně je hra přeportovaná na Unreal Engine verze 5 a k tomu nabyla konzistentního game designu a příběhu. Stejně tak byl předělán veškerý kód a architektura herní logiky. Od původní prace zbyly pouze:

- většina 3D modelů s animacemi (1/10 od aktuálního množství),
- některé UI elementy a textury (1/4 od aktuálního množství),
- level design prvního herního levelu a
- koncept herního světa a hlavní téma příběhu.

Aktuální práce značně rozvíjí příběh a přidává mnoho různých herních mechanik a systémů, grafických a zvukových assetů spolu s API pro načítání nového obsahu za běhu.

Kapitola 2

More complicated chapter

After the reader gained sufficient knowledge to understand your problem in kapitola 1, you can jump to your own advanced material and conclusions.

You will need definitions (see definice 1 below in sekce 2.1), theorems (věta 1), general mathematics, algorithms (algoritmus 1), and tables (tabulka 2.1). Obrázky 2.1 a 3.1 show how to make a nice figure. See obrázek 2.2 for an example of TikZ-based diagram. Cross-referencing helps to keep the necessary parts of the narrative close — use references to the previous chapter with theory wherever it seems that the reader could have forgotten the required context. Conversely, it is useful to add a few references to theoretical chapters that point to the sections which use the developed theory, giving the reader easy access to motivating application examples.

See documentation of package `booktabs` for hints on typesetting tables. As a main rule, *never* draw a vertical line.

2.1 Example with some mathematics

Definice 1 (Triplet)**.** *Given stuff X, Y and Z, we will write a* triplet *of the stuff as* (X, Y, Z).

Věta 1 (Car coloring)**.** *All cars have the same color. More specifically, for any set of cars C, we have*

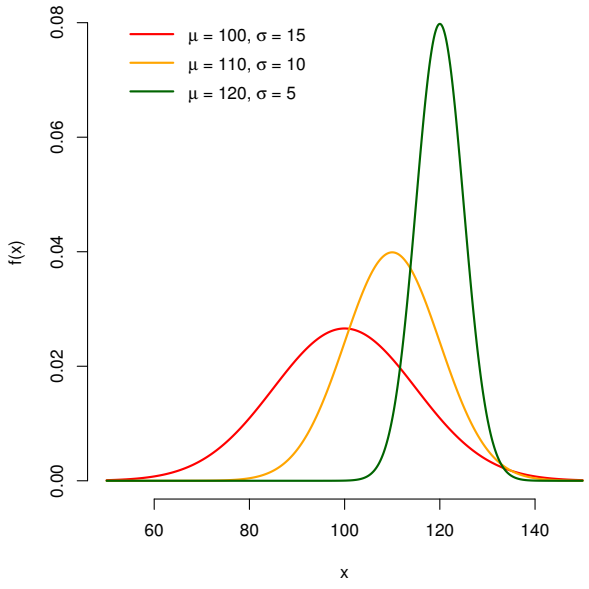
$$(\forall c_1, c_2 \in C)\ \textsc{Colour}(c_1) = \textsc{Colour}(c_2).$$

Důkaz. Use induction on sets of cars C. The statement holds trivially for $|C| \leq 1$. For larger C, select 2 overlapping subsets of C smaller than $|C|$ (thus same-colored). Overlapping cars need to have the same color as the cars outside the overlap, thus also the whole C is same-colored. □

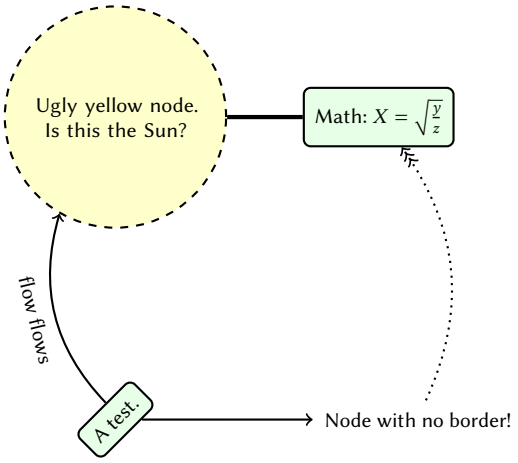
This is plain wrong though.

Column A	Column 2	Numbers	More
Asd	QWERTY	123123	–
Asd qsd 1sd	BAD	234234234	This line should be helpful.
Asd	INTERESTING	123123123	–
Asd qsd 1sd	PLAIN WEIRD	234234234	–
Asd	QWERTY	123123	–
Asd qsd 1sd	GOOD	234234299	–
Asd	NUMBER	**123123**	–
Asd qsd 1sd	DANGEROUS	234234234	(no data)

Tabulka 2.1 An example table. Table caption should clearly explain how to interpret the data in the table. Use some visual guide, such as boldface or color coding, to highlight the most important results (e.g., comparison winners).



Obrázek 2.1 A figure with a plot, not entirely related to anything. If you copy the figures from anywhere, always refer to the original author, ideally by citation (if possible). In particular, this picture — and many others, also a lot of surrounding code — was taken from the example bachelor thesis of MFF, originally created by Martin Mareš and others.



Obrázek 2.2 An example diagram typeset with TikZ. It is a good idea to write diagram captions in a way that guides the reader through the diagram. Explicitly name the object where the diagram viewing should "start". Preferably, briefly summarize the connection to the parts of the text and other diagrams or figures. (In this case, would the tenative yellow Sun be described closer in some section of the thesis? Or, would there be a figure to detail the dotted pattern of the line?)

Algoritmus 1 Algorithm that executes an action with high probability. Do not care about formal semantics in the pseudocode — semicolons, types, correct function call parameters and similar nonsense from 'realistic' languages can be safely omitted. Instead make sure that the intuition behind (and perhaps some hints about its correctness or various corner cases) can be seen as easily as possible.

function ExecuteWithHighProbability(A)
    $r \leftarrow$ a random number between 0 and 1
    $\varepsilon \leftarrow 0.00000000000000000000000000000000000000042$
    if $r \geq \varepsilon$ **then**
        execute A ..... *We discard the return value*
    else
        print: `Not today, sorry.`
    end if
end function

2.2 Extra typesetting hints

Do not overuse text formatting for highlighting various important parts of your sentences. If an idea cannot be communicated without formatting, the sentence probably needs rewriting anyway. Imagine the thesis being read aloud as a podcast — the storytellers are generally unable to speak in boldface font.

Most importantly, do not overuse bold text, which is designed to literally **shine from the page** to be the first thing that catches the eye of the reader. More precisely, use bold text only for 'navigation' elements that need to be seen and located first, such as headings, list item leads, and figure numbers.

Use underline only in dire necessity, such as in the previous paragraph where it was inevitable to ensure that the reader remembers to never typeset boldface text manually again.

Use *emphasis* to highlight the first occurrences of important terms that the reader should notice. The feeling the emphasis produces is, roughly, "Oh my — what a nicely slanted word! Surely I expect it be important for the rest of the thesis!"

Finally, never draw a vertical line, not even in a table or around figures, ever. Vertical lines outside of the figures are ugly.

Kapitola 3

Results and discussion

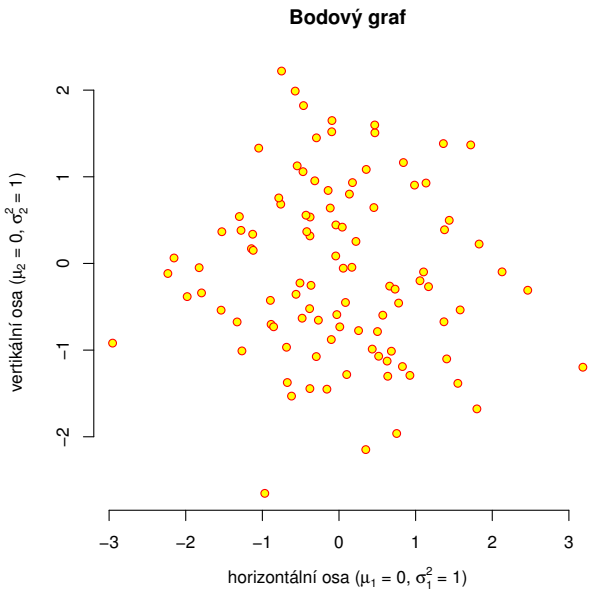
You should have a separate chapter for presenting your results (generated by the stuff described previously, in our case in kapitola 2). Remember that your work needs to be validated rigorously, and no one will believe you if you just say that 'it worked well for you'.

Instead, try some of the following:

- State a hypothesis and prove it statistically
- Show plots with measurements that you did to prove your results (e.g. speedup). Use either `R` and `ggplot`, or Python with `matplotlib` to generate the plots.[1] Save them as PDF to avoid printing pixels (as in obrázek 3.1).
- Compare with other similar software/theses/authors/results, if possible
- Show example source code (e.g. for demonstrating how easily your results can be used)
- Include a 'toy problem' for demonstrating the basic functionality of your approach and detail all important properties and results on that
- Include clear pictures of 'inputs' and 'outputs' of all your algorithms, if applicable

It is sometimes convenient (even recommended by some journals, including Cell) to name the results sub-sections so that they state what exactly has been achieved. Examples follow.

[1] Honestly, the plots from `ggplot` look much better.



Obrázek 3.1 This caption is a friendly reminder to never insert figures "in text," without a floating environment, unless explicitly needed for maintaining the text flow (e.g., the figure is small and developing with the text, like some of the centered equations, as in věta 1). All figures *must* be referenced by number from the text (so that the readers can find them when they read the text) and properly captioned (so that the readers can interpret the figure even if they look at it before reading the text — reviewers love to do that).

3.1 SuperProgram is faster than OldAlgorithm

3.1.1 Scalability estimation

3.1.2 Precision of the results

3.2 Weird theorem is proven by induction

3.3 Amount of code reduced by CodeRedTool

3.3.1 Example

3.3.2 Performance on real codebases

3.4 NeuroticHelper improves neural network learning

3.5 Graphics and figure quality

No matter how great the text content of your thesis is, the pictures will always catch the attention first. This creates the very important first impression of the thesis contents and general quality. Crucially, that also decides whether the thesis is later read with joy, or carefully examined with suspicion.

Preparing your thesis in a way such that this first impression gets communicated smoothly and precisely helps both the reviewer and you: the reviewer will not have a hard time understanding what exactly you wanted to convey, and you will get a better grade.

Making the graphics 'work for you' involves doing some extra work that is often unexpected. At the same time, you will need to fit into graphics quality constraints and guidelines that are rarely understood before you actually see a bad example. As a rule of thumb, you should allocate at least the same amount of time and effort for making the figures look good as you would for writing, editing and correcting the same page area of paragraph text.

3.5.1 Visualize all important ideas

The set of figures in your thesis should be comprehensive and complete. For all important ideas, constructions, complicated setups and results there should be a visualization that the reader can refer to in case the text does not paint the 'mental image' sufficiently well. At the bare minimum, you should have at least 3

figures (roughly corresponding to the 3 chapters) that clearly and unambiguously show:

1. the context of the problem you are solving, optionally with e.g. question marks and exclamation marks placed to highlight the problems and research questions

2. the overall architecture of your solution (usually as a diagram with arrows, such as in obrázek 2.2, ideally with tiny toy examples of the inputs and outputs of each box),

3. the advancement or the distinctive property of your solution, usually in a benchmark plot, or as a clear demonstration and comparison of your results.

3.5.2 Make the figures comprehensible

The figures should be easily comprehensible. Surprisingly, that requires you to follow some common "standards" in figure design and processing. People are often used to a certain form of the visualizations, and (unless you have a very good reason) deviating from the standard is going to make the comprehension much more complicated. The common standards include the following:

- caption everything correctly, place the caption at an expectable position

- systematically label the plots with 'main' titles (usually in boldface, above the plot), plot axes, axis units and ticks, and legends

- lay out the diagrams systematically, ideally follow a structure of a bottom-up tree, a left-to-right pipeline, a top-down layered architecture, or a center-to-borders mindmap

- use colors that convey the required information correctly

  Although many people carry some intuition for color use, achieving a really correct utilization of colors is often very hard without previous experience in color science and typesetting. Always remember that everyone perceives color hues differently, therefore the best distinction between the colors is done by varying lightness of the graphics elements (i.e., separating the data by dark vs. light) rather than by using hues (i.e., forcing people to guess which one of salmon and olive colors means "better"). Almost 10% of the population have their vision impaired by some form of color vision deficiency, most frequently by deuteranomaly that prevents interpretation of even the most 'obvious' hue differences, such as green vs. red. Finally, printed colors look surprisingly different from the on-screen

colors. You can prevent much of these problems by using standardized palettes and well-tested color gradients, such as the ones from ColorBrewer[2] and ViridisLite[3]. Check if your pictures still look good if converted to greyscale, and use a color deficiency simulator to check how the colors are perceived with deuteranomaly.

Avoid large areas of over-saturated and dark colors:

- under no circumstances use dark backgrounds for any graphical elements, such as diagram boxes and tables — use very light, slightly desaturated colors instead
- avoid using figures that contain lots of dark color (as a common example, heatmaps rendered with the 'magma' color palette often look like huge black slabs that are visible even through the paper sheet, thus making a dark smudge on the neighboring page)
- increase the brightness of any photos to match the average brightness of the text around the figure

Remember to test your figures on other people — usually, just asking 'What do you think the figure should show?' can help you debug many mistakes in your graphics. If they think that the figure says something different than what you planned, then most likely it is your figure what is wrong, not the understanding of others.

Finally, there are many magnificent resources that help you arrange your graphics correctly. The two books by Tufte [6, 7] are arguably classics in the area. Additionally, you may find many interesting resources to help you with technical aspects of plotting, such as the `ggplot`-style 'Fundamentals' book by Wilke [8], and a wonderful manual for the TikZ/PGF graphics system by Tantau [9] that will help you draw high-quality diagrams (like the one in obrázek 2.2).

3.6 What is a discussion?

After you present the results and show that your contributions work, it is important to *interpret* them, showing what they mean in the wider context of the thesis topic, for the researchers who work in the area, and for the more general public, such as for the users.

Separate discussion sections are therefore common in life sciences where some ambiguity in result interpretation is common, and the carefully developed intuition about the wider context is sometimes the only thing that the authors

[2] `https://colorbrewer2.org`

[3] `https://sjmgarnier.github.io/viridisLite/`

have. Exact sciences and mathematicians do not need to use the discussion sections as often. Despite of that, it is nice to position your output into the previously existing environment, answering:

- What is the potential application of the result?
- Does the result solve a problem that other people encountered?
- Did the results point to any new (surprising) facts?
- How (and why) is the approach you chose different from what the others have done previously?
- Why is the result important for your future work (or work of anyone other)?
- Can the results be used to replace (and improve) anything that is used currently?

If you do not know the answers, you may want to ask the supervisor. Also, do not worry if the discussion section is half-empty or thoroughly pointless; you may remove it completely without much consequence. It is just a bachelor thesis, not a world-saving avenger thesis.

Conclusion

In the conclusion, you should summarize what was achieved by the thesis. In a few paragraphs, try to answer the following:

- Was the problem stated in the introduction solved? (Ideally include a list of successfully achieved goals.)
- What is the quality of the result? Is the problem solved for good and the mankind does not need to ever think about it again, or just partially improved upon? (Is the incompleteness caused by overwhelming problem complexity that would be out of thesis scope, or any theoretical reasons, such as computational hardness?)
- Does the result have any practical applications that improve upon something realistic?
- Is there any good future development or research direction that could further improve the results of this thesis? (This is often summarized in a separate subsection called 'Future work'.)

This is quite common.

Seznam použité literatury

[1] J. Schreier. *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made*. HarperCollins, 2017. ISBN: 9780062651242. URL: https://books.google.cz/books?id=-bK-DQAAQBAJ.

[2] Mike Lopez. *Gameplay Fundamentals Revisited: Harnessed Pacing & Intensity [online]*. [cit. 2025-04-01]. Dostupné z: https://www.gamedeveloper.com/design/gameplay-fundamentals-revisited-harnessed-pacing-intensity. 2018.

[3] Sirawat Pitaksarit. *Objectively comparing Unity and Unreal Engine*. [cit. 2025-04-01]. Dostupné z: https://gametorrahod.com/objectively-comparing-unity-and-unreal-engine/. 2019.

[4] Reddit/Godot. *What are the worst parts/issues of Godot in your experience?* [cit. 2025-04-01]. Dostupné z: https://www.reddit.com/r/godot/comments/1ewrkey/what_are_the_worst_partsissues_of_godot_in_your/. 2024.

[5] Alfred Reinold Baudisch. *Godot's 3D Workflow Issues, Inconsistencies, and Confusion*. [cit. 2025-04-01]. Dostupné z: https://alfredbaudisch.com/blog/gamedev/godot-engine/godots-3d-confusing-workflow-inconsistencies-conflicting-behaviours-and-annoyances/. 2023.

[6] Edward R Tufte, Nora Hillman Goeler a Richard Benson. *Envisioning information*. Graphics press Cheshire, CT, 1990.

[7] Edward R Tufte. *Visual display of quantitative information*. Graphics press Cheshire, CT, 1983.

[8] Claus O Wilke. *Fundamentals of Data Visualization*. O'Reilly Media, Inc., 2019. ISBN: 9781492031086. URL: https://clauswilke.com/dataviz/.

[9] Till Tantau. *The TikZ and PGF Packages (Manual for version 3.1.8b)*. Tech. zpr. Institut für Theoretische Informatik Universität zu Lübeck, 2020. URL: http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf.

Příloha A

Using CoolThesisSoftware

Use this appendix to tell the readers (specifically the reviewer) how to use your software. A very reduced example follows; expand as necessary. Description of the program usage (e.g., how to process some example data) should be included as well.

To compile and run the software, you need dependencies XXX and YYY and a C compiler. On Debian-based Linux systems (such as Ubuntu), you may install these dependencies with APT:

```
apt-get install \
  libsuperdependency-dev \
  libanotherdependency-dev \
  build-essential
```

To unpack and compile the software, proceed as follows:

```
unzip coolsoft.zip
cd coolsoft
./configure
make
```

The program can be used as a C++ library, the simplest use is demonstrated in výpis 1. A demonstration program that processes demonstration data is available in directory `demo/`, you can run the program on a demonstration dataset as follows:

```
cd demo/
./bin/cool_process_data data/demo1
```

After the program starts, control the data avenger with standard `WSAD` controls.

Výpis kódu 1 Example program.

```
#include <CoolSoft.h>
#include <iostream>

int main() {
  int i;
  if(i = cool::ProcessAllData()) // returns 0 on error
    std::cout << i << std::endl;
  else
    std::cerr << "error!" << std::endl;
  return 0;
}
```