Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Edukativní aplikace s využitím Unreal Engine

Bakalářská práce

Autor: Petr Vavřínek

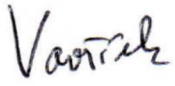
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové duben 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.



V Hradci Králové dne 28.4.2023 Petr Vavřínek

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Daniele Ponce, Ph.D. za metodické vedení a jakoukoliv pomoc při zpracování této práce.

Anotace

Tato bakalářská práce přibližuje čtenáři pojmy jako herní engine, edukativní aplikace a další spojené technologie či pojmy s vývojem her. Vysvětluje, jaký je proces vývoje her, co jsou to herní enginy se zaměřením na Unreal Engine. Poukazuje na vizuální programování jeho výhody a nevýhody. Je zde i část, kdy se práce věnuje využití počítačových her ve vzdělání a jeho dopady na rozvoj, ať už pozitivní tak i negativní. V praktické části je využit Unreal Engine pro vytvoření vzdělávací aplikace zaměřené na logické myšlení právě při vývoji her. V této části je demonstrován návrh aplikace vytváření herních funkcionalit uživatelského rozhraní a dalších částí s tím spojených.

Annotation

Title: Educational application using Unreal Engine

This bachelor thesis introduces the reader to concepts such as game engines, educational applications, and other related technologies or terms in game development. It explains the game development process and focuses on game engines, particularly the Unreal Engine. It highlights the advantages and disadvantages of visual programming. There is also a section dedicated to the use of computer games in education and their impact on development, both positive and negative. In the practical part, Unreal Engine is used to create an educational application that focuses on logical thinking during game development. This section demonstrates the design of the application for creating game functionality, user interface, and other related parts.

Obsah

1. Úvod

V dnešní době jsou počítačové hry stále populárnější a nacházejí své uplatnění nejen v zábavě, ale také v edukaci. Tato práce se zabývá využitím herního enginu pro vývoj edukativních aplikací, a to konkrétně Unreal Engine. Cílem práce je zkráceně vytvoření funkční vzdělávací aplikace, která zlepšuje logické myšlení v oblasti programování. Tato problematika bude dále nastíněna v následující kapitole. Kapitola metodiky se zabývá postupným rozdělením problému na dílčí části, jejich seřazením a následným vykonáním. Práce se zabývá samotným vývojem her, poukazuje na jeho proces a obeznamuje čtenáře o jednotlivých postupech při vývoji.

Jelikož většina dnešních her není postavena od základů, je zde vysvětlen pojem herní engine se zaměřením na Unreal Engine, jeho části a vybraná funkcionalita pro porozumění praktické části. Čtenář je obeznámen se samotným pojmem *Vizuální programování.* Důvodem, proč se dnes používá, dnešním využitím, pozitivy a negativy tohoto přístupu k programování. Předmětem práce jsou edukativní počítačové hry, je zde zmíněna jejich definice, přínosy a negativa pro uživatele.

Praktická část je zaměřena na vývoj edukativní aplikace s využitím Unreal Engine 5. V této části je zmíněno rozvržení, tj. analýza, návrh a následně samotná implementace, kde jsou demonstrovány části vývoje aplikace. V analytické části je uvedena idea projektu a funkcionalit, které by měla výsledná aplikace splňovat. Praktická část je věnována jednotlivým částem vývoje hry, je zde zmíněno vytváření herních prvků, rozplánování jednotlivých funkcionalit a následná implementace. Součástí implementační části je také vysvětlení určitých herních mechanik, které byly v projektu použity.

2. Cíl práce

Cílem této práce je vytvoření vzdělávací hry v Unreal Enginu. Samotná hra bude zaměřena hlavně na rozvoj v oblasti logického myšlení. Rovněž budou rozebrány jednotlivé kroky procesu vývoje her, včetně návrhu, implementace, testování a publikování. stejně jako klíčové prvky herních enginů a vizuálního programování. Čtenář bude seznámen s herním enginem Unreal Engine a pojmem vizuální programování, a to rámci jak teoretické, tak i praktické části. Bude objasněno, co je to engine a jaký je rozdíl mezi ním a samotným editorem. Též zde budou zmíněné vybrané funkcionality použité v praktické části práce. Tedy výsledkem této práce bude ucelené pochopení procesu vývoje her, včetně použití herních enginů a vizuálního programování a praktická aplikace těchto poznatků v podobě vytvořené vzdělávací aplikace.

3. Metodika zpracování

Metodika zpracování aplikace zahrnuje několik klíčových kroků, které byly provedeny během vývoje aplikace. Následuje stručný popis jednotlivých fází:

1. Analýza požadavků a specifikace aplikace: V této fázi byly identifikovány požadavky na aplikaci. Byla provedena analýza a specifikace funkcionalit, uživatelského rozhraní, technických požadavků a dalších relevantních aspektů.
2. Výběr technologií a nástrojů pro vývoj: Byly vybrány vhodné technologie a nástroje pro vývoj aplikace na základě požadavků a cílů projektu. Mezi použité technologie patří například herní enginy, programovací jazyky, vývojová prostředí atd.
3. Implementace a programování aplikace: Byl proveden vývoj aplikace na základě navržených specifikací. Byl použit iterativní vývojový proces, který zahrnoval programování funkcionalit, tvorbu herních objektů atd.

Odpovědi na otázky byly ve většině případů nalezeny z internetových zdrojů jako oficiální dokumentace a dalších materiálů zabývajících se podobným tématem.

4. Vývoj her

Vývoj her je proces vytváření počítačových her od jejich konceptu a návrhu až po konečnou implementaci a vydání. Tento proces zahrnuje mnoho různých aspektů, včetně programování, designu, umění, testování a marketingu.

4.1. Proces vývoje hry

Vytváření počítačových her je složitý proces, který obvykle zahrnuje několik kroků. Tyto kroky se mohou lišit v závislosti na specifickém projektu a týmu vývojářů, ale v obecnosti mohou zahrnovat následující procesy:

4.1.1. Návrh

Tento krok zahrnuje definování cílů hry, příběhu, herního světa, herních pravidel a dalších klíčových prvků. Tento proces může zahrnovat tvorbu konceptů, storyboardů a prototypů.

Koncept obvykle používá pro abstraktní modely, které popisují určitou oblast problému. Koncepty se používají k návrhu a vývoji softwaru, aby bylo možné efektivněji a jednodušeji řešit problémy. V programování mohou být třeba objektově orientované programování, návrhové vzory nebo architektury softwaru.

Prototyp je zpravidla reprezentace části kódu nebo softwarového systému, který slouží k testování a ověřování funkcí a vlastností předtím, než jsou integrovány do finální verze. (MEMOS)

Storyboard je vizuální reprezentace příběhu nebo scénáře v podobě série ilustrací nebo snímků, které popisují postupné dění v určitém procesu nebo situaci. V oblasti informačních technologií se storyboard používá pro návrh uživatelského rozhraní, návrh webů, her a dalších aplikací. (Pařízek)

4.1.2. Implementace

V této části vzniká už něco *interaktivního*. Vytvářejí se 2D návrhy, 3D modely, textury, uživatelské rozhraní a programuje se funkcionalita dílčích částí.

Právě v této části tvoření hry se nejčastěji používají herní enginy, které jsou rozebrány níže. (Bramble, 2023)

4.1.3. Testování

Před finálním publikováním je nutné zkontrolovat, zda všechny funkcionality vyhovují zadání. Pokud se tak nestane, je nutné se vrátit o krok zpět, opravit zjištěné chyby a opět začít testovat. Tento proces se opakuje do doby, kdy je s ním pověřená osoba spokojena. S tímto je spojeno i testování obtížností, chyb v herním světě apod. (Bramble, 2023)

4.1.4. Publikování

V případě, že chce uživatel hru publikovat sám, je to možné provést přes různé platformy jako je *Steam*, méně známé *Itch.io*, *Epic Games*, atd. Na trhu je spousta firem, které se zabývají pouze publikací a marketingem her. Tyto firmy jsou nazývány herními vydavateli a jejich náplní práce je přivést hru na trh a učinit ji úspěšnou, popř. zařídit, aby se tato hra dostala do povědomí lidí pomocí marketingu. (Indeed Editorial Team, 2022)

4.2. Herní engine

Zkráceně, je to software, který pomáhá, či zjednodušuje vytvoření dalšího softwaru v oblasti videoher. Samotný engine je většinou skládán z dílčích částí, které spolupracují a tvoří jeden velký funkční celek.

„Engine umí vykreslovat grafiku, vypočítávat fyziku, pracovat s nasvícením, zvukem a podobně. Zároveň výrazně pomáhají s převodem hry na různé platformy. Herních enginů je nespočet a každý má daná pravidla i systém fungování, který vývojářům hry usnadňuje tvorbu, jakmile se s ním naučí pracovat. S nadsázkou lze říct, že jde o jakýsi předpřipravený herní korpus, na který se přidávají jednotlivé vlastní ingredience. Často však enginy zároveň mívají knihovnu předpřipravených prvků (např. skeny terénu, univerzální 3D modely), které lze využít.“ (Pešek, 2022)

4.3. Dnešní herní engine

Dnes existuje několik jak proprietárních, tak i veřejně dostupných herních enginů. Každý má trochu jinou architekturu, nelze tedy předpokládat, že hry se v nich budou vyvíjet stejně, či podobně. Mezi dnes nejrozšířenější se řadí Unreal Engine, Unity, Godot nebo třeba CryEngine.

Dle (Merheb, 2023) v tomto výběru vyhrává Unity engine a Unreal engine je v těsném závěru. Autor hodnocení (Merheb, 2023) zvážil určité aspekty, následně porovnal dle vlastního hodnocení. Do něj se řadily funkcionality, uživatelská přívětivost, cena a náročnost k používání.

4.4. Unreal Engine

Dle oficiální stránky Unreal Engine, se jedná o kompletní sadu nástrojů pro vývoj her, architektonickou a automobilovou vizualizaci, tvorbu filmového a televizního obsahu, vysílání a produkci živých akcí, školení, simulace a další aplikace. (Unreal Engine) Zkráceně to je tedy software pro tvoření dalšího softwaru, avšak už jednodušeji. Vyšel 19. března roku 2014 na *Game Developers Conference*, což je konference o videoherním průmyslu kde firmy představují své produkty. V práci dále označován jako UE.

4.4.1. Editor, engine a moduly

Editor je první věc, kterou si uživatel vybaví, a právě v něm vzniká celý projekt. Nicméně samotný editor je pouze součást uživatelského grafického rozhraní, skrze který uživatel komunikuje s logickou vrstvou na nižší úrovni. Tato logická vrstva je již samotný engine, který spojuje jednotlivé moduly do jednoho celku. Editor tedy uživateli přináší jednoduché ovládání a orientaci v projektu.

4.4.2. Engine

Celý engine je tvořen z modulů, které mají vazby mezi sebou. Dle oficiální stránky jsou moduly základním stavebním blokem architektury Unreal Enginu. Každý modul může obsahovat nástroje pro editor, funkce, knihovny a další.

Jádro Unreal Engine se skládá z následujících modulů:

- Modul jádra enginu (Core): Poskytuje základní funkce pro správu paměti, souborového systému, vláken a další základní funkce, které jsou potřebné pro běh enginu.
- Modul herního světa (World): Zajišťuje správu virtuálního herního světa, včetně správy objektů, kamer a osvětlení.
- Modul renderování (Renderer): Zodpovědný za vytváření vizuálního výstupu enginu, včetně výpočtu stínů, reflexí a dalších vizuálních efektů.
- Modul fyzikální simulace (Physics): Obsahuje simulaci fyzikálních objektů a kolizí mezi nimi.

Mimo tyto také obsahuje další jako jsou (Unreal Engine)

- Modul umělé inteligence (AI): Poskytuje nástroje pro vytváření inteligentních herních agentů.
- Modul zvukových efektů (Audio): Obsahuje nástroje pro vytváření a správu zvukových efektů.
- Modul síťových funkcí (Networking): Poskytuje nástroje pro vytváření a správu multiplayerových her.

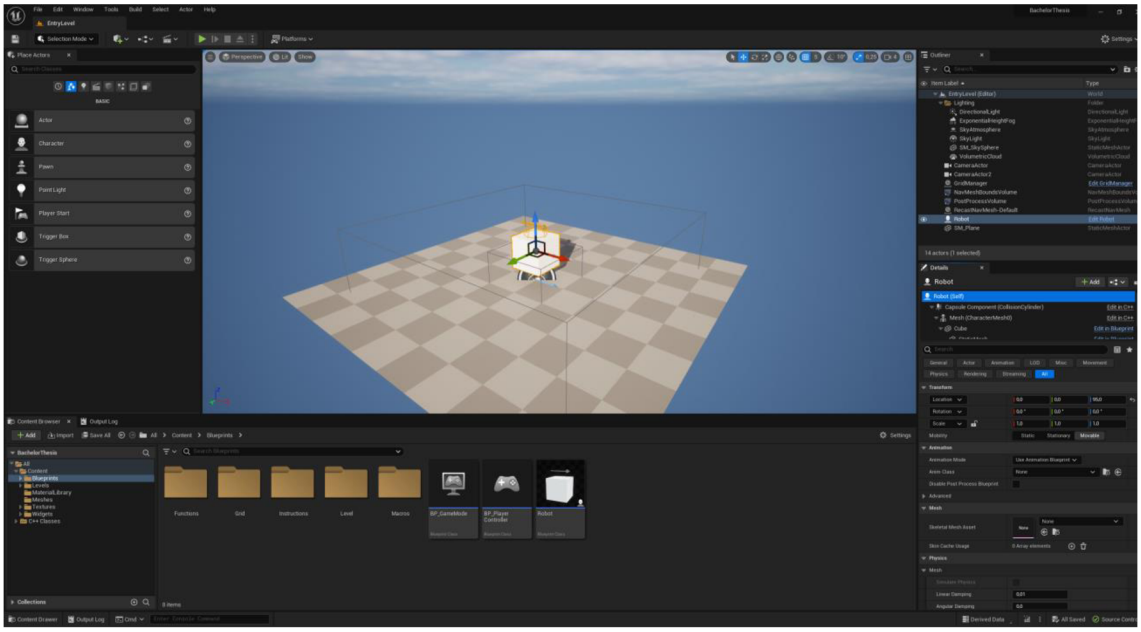
Též dle oficiálního zdroje. (Unreal Engine) přináší tento systém následující výhody:

- Moduly zajišťují dobré oddělení kódu a poskytují prostředky pro zapouzdření funkcí a skrytí vnitřních částí kódu.
- Moduly jsou kompilovány jako samostatné kompilační jednotky. To znamená, že je třeba kompilovat pouze moduly, které se změnily, a doba sestavení větších projektů se výrazně zkrátí.
- Lze manuálně řídit kdy má být modul odpojen nebo připojen, a to i za běhu. To umožňuje optimalizovat výkon projektu.

Jednou z dalších velice důležitých výhod je jednoduchost v přidávání vlastních zásuvných modulů, tzv. pluginů. Jedná se o typ modulu, který nemusí být součástí celého enginu, lze jej přidat pouze k danému projektu.

4.4.3. Editor

Jak již bylo zmíněno, editor je prostředek k tvoření her, či grafických aplikací v grafické podobě. Editor obsahuje grafické nástroje pro kompletní vývoj her. Lze zde najít prohlížeč herních souborů, tzv. *Content Browser*, herní scény *Viewport* a další panely. Vzhled editoru je plně modulární, uživatel si tedy může prostředí přizpůsobit svým potřebám.



Obrázek 1 Unreal Engine 5 – Editor při vývoji praktické části aplikace (zdroj: vlastní)

Jak již bylo řečeno v předchozím odstavci, funkce editoru jsou silně spjaty s načtenými moduly v enginu. Díky nim následně přibývají další funkce, panely a podobná rozšíření.

Editor sám o sobě je též modulem, stejnojmenným. Pro kompletní funkcionalitu jsou potřeba další moduly jako jsou např.:

- Modul pro tvorbu obsahu (Content Browser): Umožňuje vývojářům procházet a organizovat herní obsah, jako jsou modely postav, textury a zvukové efekty.
- Modul pro úpravy herního světa (World Editor): Obsahuje nástroje pro úpravu virtuálního herního světa, jako jsou terén, osvětlení, kamery a další prvky.
- Modul pro tvorbu postav (Persona): Poskytuje nástroje pro tvorbu animací postav.
- Modul pro vizualizaci (dříve Cascade, dnes Niagara): Umožňuje vývojářům vytvářet vizuální efekty, jako jsou ohně, exploze a další efekty.
- Modul pro tvorbu UI (UMG): Poskytuje nástroje pro tvorbu uživatelského rozhraní.

4.4.4. Proces získání enginu

Jako první krok bude registrace uživatele a následně jsou dvě možnosti.
Pokud uživateli stačí základní verze bez zdrojového kódu, popř. pouze C++ hlavičky, může použít *Epic Games Launcher*, software od Epic Games, který umožňuje stáhnout UE od verze 4.0.2 až do 5.x.
Když si uživatel bude chtít zkompilovat celý projekt sám, musí vlastnit GitHub účet a následně ho propojit se svým Epic Games účtem a tím souhlasit s podmínkami Epic Games. Poté by uživatel měl mít přístup do UnrealEngine repozitáře, ve kterém si může zvolit jakoukoliv verzi nebo dokonce verzi, kde byla udělána jakákoliv změna po celou dobu existence tohoto repozitáře. K samotnému zkompilování by uživatel měl použít oficiální dokumentaci, která se časem může měnit.

4.4.5. Vybraná funkcionalita

Praktická část bakalářské práce využívá několik pojmů, které se v oblasti UE vyskytují, následné odstavce tedy budou věnovány definicím a pojmech v UE.

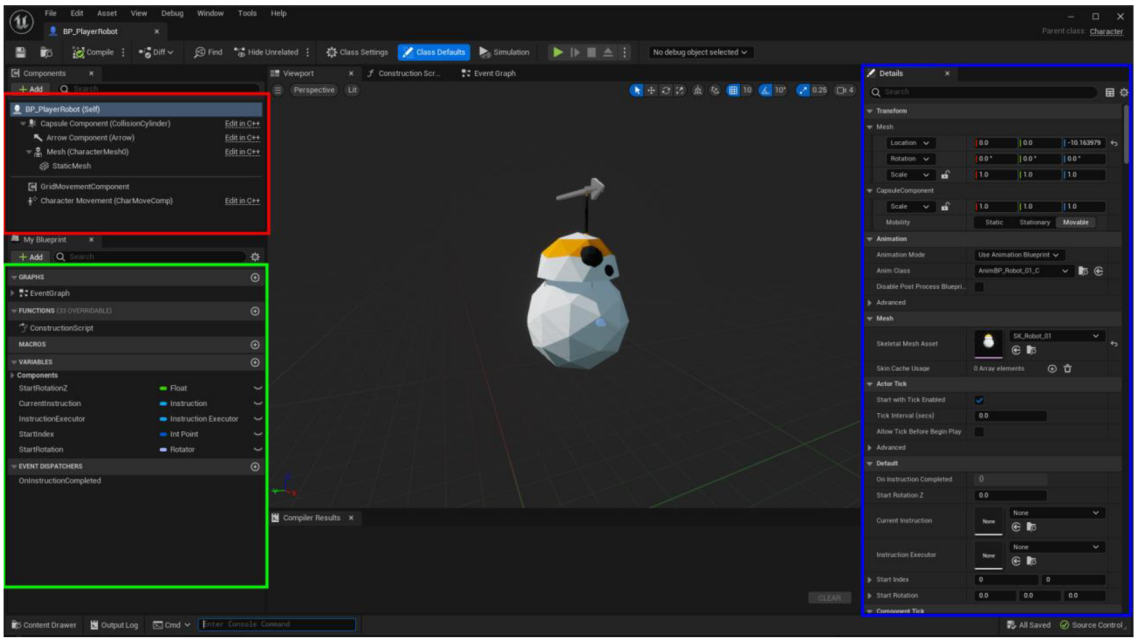
4.4.5.1. Třída Object

Třída Object je základní třídou UE. Samotnou implementaci této třídy lze upravovat v blueprintech. Objekty na základě této třídy jsou obohaceny podporou *Garbage collection*. Tato funkcionalita je velice důležitá pro chod celé aplikace. Zařizuje, aby byla průběžně mazána data v operační paměti z důvodu jejich nevyužití.

Díky těmto výhodám, tuto třídu dědí většina tříd v celém enginu.

4.4.5.2. Třída Actor

Tato třída je speciální v tom, že může existovat v herním světě. Dědí přímo ze třídy Object, takže je obohacena o všechny její vlastnosti.



Obrázek 2 Unreal Engine 5 – Editor – Actor (zdroj: vlastní)

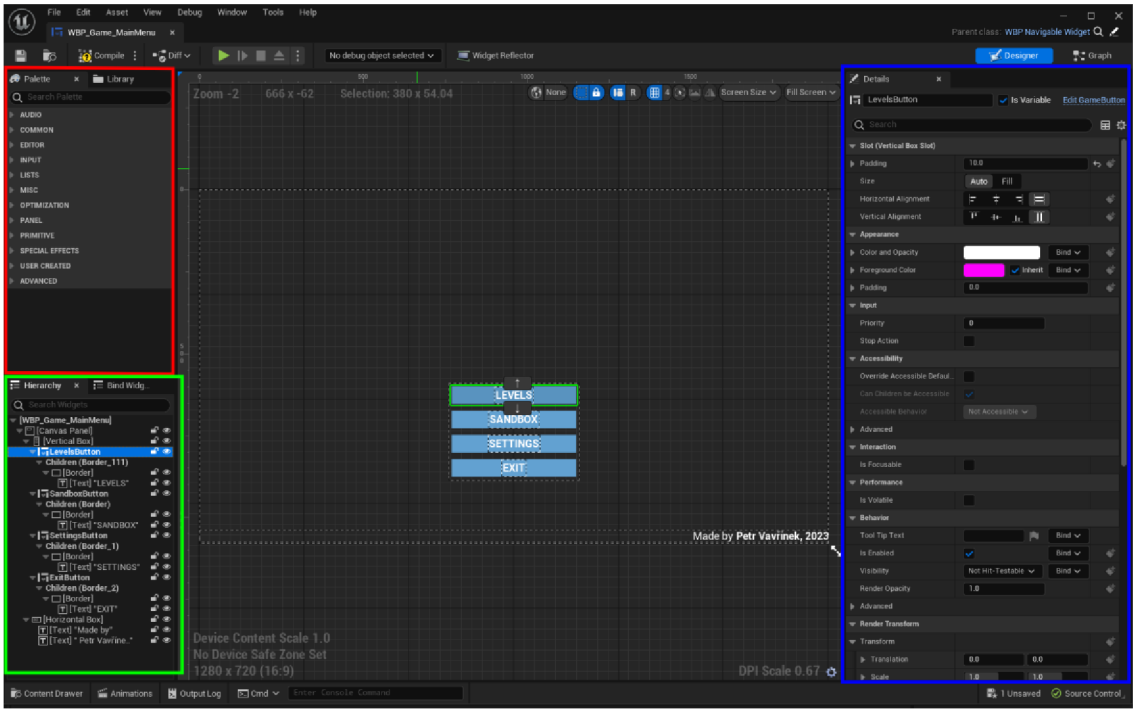
Na obrázku lze vidět několik různých částí. V červeném obdélníku se nachází tzv. komponenty, které jsou vázány na třídu. Umožňují přidávat meshe (3D model) do třídy, rozpohybovat daný objekt, či přidávat další dodatečnou funkcionalitu.

V zeleném obdélníku jsou vidět uživatelem definované funkce, makra, proměnné apod.

V oblasti modrého jsou detaily třídy, typicky se jedná o nastavení výchozích hodnot, tato oblast se mění dle zvoleného prvku v UI.

4.4.5.3. Widget a widget editor

Hry obsahují jak 3D scénu, tak i další vrstvy, které překrývají části obrazu.
V UE se nazývají widgety. Hlavním úkolem je rozšíření uživatelského rozhraní
o interaktivní prvky či pouze zobrazovat určité informace v grafické podobě.



Obrázek 3 Unreal Engine 5 – Widget editor (zdroj: vlastní)

Proces tvoření rozhraní spočívá v *drag and drop* mechanice, kdy uživatel tahá myší elementy v červeném boxu, z tzv. palety do okna. Následně se element objeví v hierarchii elementů (zelený obdélník). Jednotlivé elementy lze zanořovat do sebe, ale pouze v případě, že element tuto funkcionalitu podporuje. V modrém obdélníku se objevují detaily zvoleného elementu v hierarchii, pod tímto si může uživatel představit např. styl písma, barvu pozadí apod.

4.4.5.4. PlayerController Actor

PlayerController je rozhraní mezi postavou a lidským hráčem, který ho ovládá. V podstatě představuje vůli lidského hráče. (Unreal Engine)
Pomocí tohoto objektu lze měnit ovládání herních postav za běhu hry. V Unreal Engine se tento proces nazývá *Possess* a *Unpossess*. Na rozdíl od postavy hráče přináší jistou persistenci dat mezi nimi. Např.: když naše postava umře nebo se smaže z herního světa, tak jsou veškeré data, které postava obsahovala ztracené.

4.4.5.5. GameMode Actor

Hra má určitá pravidla, která tvoří herní režim. Na nejzákladnější úrovni tato pravidla zahrnují (Unreal Engine):

- Počet přítomných hráčů, diváků a jejich maximální povolený počet.
- Způsob, jakým hráči vstupují do hry, což může zahrnovat pravidla pro výběr míst spawnu a další chování při spawnu nebo respawnu.
  Spawn – vytvoření například postavy v herním světě.
  Respawn – Opakovaný spawn.
- Zda je možné hru pozastavit a jak je pozastavení hry řešeno.
- Přechody mezi úrovněmi, včetně toho, zda má hra začínat ve filmovém režimu.

Více lze najít v oficiální dokumentaci.

4.4.5.6. Pawn Actor

Dle (Unreal Engine) je Pawn základní třídou všech Actor objektů, které mohou ovládat hráči nebo umělá inteligence. Je to fyzická reprezentace hráče nebo umělé inteligence v herním světě. To znamená, že Pawn určuje nejen to, jak hráč vypadá vizuálně, ale také to, jak interaguje s herním světem z hlediska kolizí a dalších fyzických interakcí.

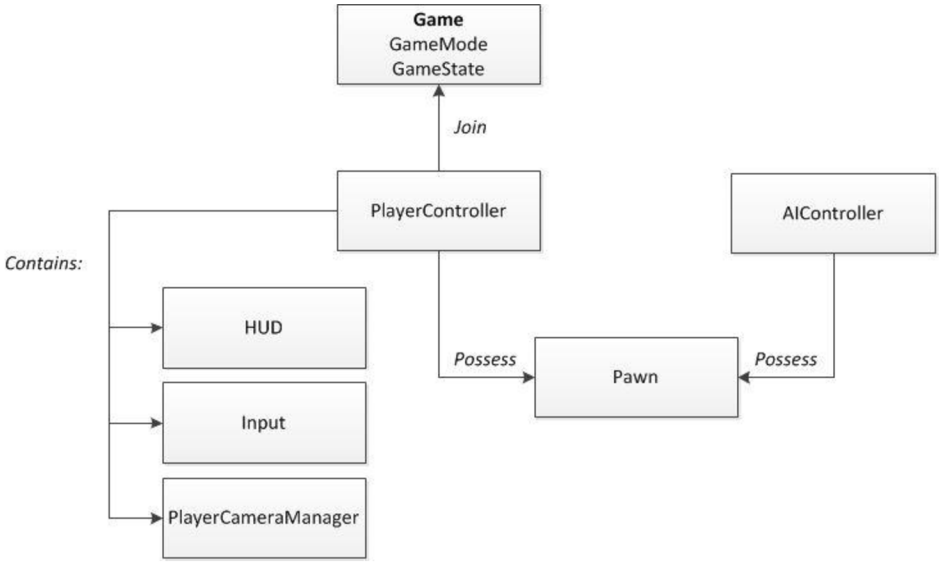
4.4.5.7. Propojení GameMode, PlayerController a Pawn tříd

V každém herním světě se stará o chod hry GameMode objekt. Nachází se v něm kontrolované postavy třídy Pawn, které jsou kontrolovány buďto hráčem pomocí PlayerController objektu nebo umělou inteligencí pomocí AIControlleru.

Každý hráč má vlastní HUD, PlayerCameraController a další které jsou potřebné pro obstarání veškerých herních funkcionalit.

HUD je *heads-up displej* neboli 2D displej na obrazovce, který je běžný v mnoha hrách. Typicky se může jednat o zobrazení munice, zdraví atd.

Dle (Unreal Engine) je PlayerCameraManager "oční bulvou" hráče a řídí chování kamery.



Obrázek 4 Unreal Engine – Propojení GameMode, PlayerController a Pawn tříd (zdroj: docs.unrealengine.com)

4.4.5.8. Zveřejnění vytvořeného produktu

Před samotným zveřejněním je nutné aplikaci dostat ze stavu projektu do zkompilované aplikace pro danou platformu. Unreal Engine podporuje platformy jako je Android, Windows, Linux, iOS a další. K tomu, aby uživatel mohl hru zkompilovat, budou potřeba odpovídající knihovny pro danou platformu. Poté, co bude projekt zkompilován, se vytvoří projektová struktura včetně spustitelné aplikace. Tu lze následně buďto spustit, či nahrát na platformy jako je Google Play, Steam apod.

5. Vizuální programování

Vizuální programování je způsob vývoje aplikací, při kterém uživatel nemusí znát konkrétní syntaxi programovacího jazyka, který nemusí být ani specifikován, jelikož nepoužívá k programování textový režim. Jde tedy o nějaké abstraktní programování na nejvyšší možné úrovni. Každá aplikace, která používá vizuální programování si definuje vlastní pravidla a vzhled.

K samotnému programování je využito grafické znázornění bloků, či jiných obrazců, které provádí určitou akci. Vzhled a proces programování tímto stylem je silně spjat s daným software, který je vždy jiný. Neexistuje tedy žádný standard, který by všechna vývojářská prostředí implementovala. Z toho plyne, že při používání různých vývojových prostředí neexistuje jednotný postup.

5.1. Důvody proč se používá

Jak již bylo zmíněno uživatel nepotřebuje znát konkrétní syntaxi názvy funkcí a jediné, co mu stačí, je grafické rozhraní, kde si vše obstará pomocí vizuálních bloků, čar a dalších obrazců, které laikovi dávají větší smysl. To nutně neznamená, že uživatel se bude ihned v prostředí orientovat, ale určitě se ho bude schopen rychleji naučit ovládat.

5.2. Stinné stránky

Nevýhody vizuálního programování se liší v závislosti na konkrétním nástroji nebo platformě, kterou uživatel používá, ale zde je několik obecných nevýhod, které se často uvádějí:

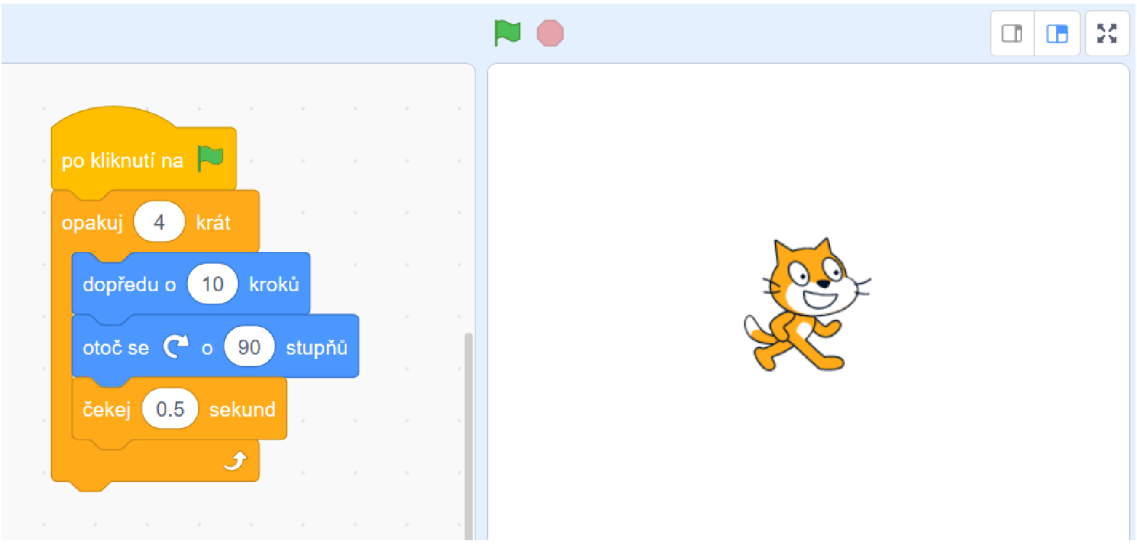
1. Omezená flexibilita: Vizuální programování může být méně flexibilní než textové programování, protože závisí na předdefinovaných blocích nebo prvcích, které jsou poskytovány v grafickém rozhraní. To může omezovat schopnost provádět pokročilé úkoly nebo implementovat složitější logiku, kterou lze dosáhnout s textovým programováním.
2. Omezená rozšiřitelnost: Vizuální programování může mít omezenou rozšiřitelnost oproti textovému programování, protože může být obtížné nebo nemožné přidat vlastní funkce nebo moduly do vizuálního rozhraní. To může omezovat schopnost používat externí knihovny nebo rozšíření a omezovat potenciál pro pokročilé a specifické úkoly. (Huivan, 2022)
3. Složitější ladění a debuggování: Vizuální programování může být obtížné, pokud jde o ladění a debuggování chyb. Vizualizace programu může být abstraktní a složitější identifikovat a opravit chyby v porovnání s textovým programováním, které poskytuje podrobnější a jasnější chybové zprávy. (Ossian, 2022)
4. Omezená kompatibilita a přenositelnost: Programy vytvořené pomocí vizuálního programování mohou být omezené v kompatibilitě a přenositelnosti na jiné platformy nebo systémy. To může omezovat schopnost sdílet nebo přenášet kód mezi různými prostředími nebo zařízeními.
5. Přehlednost a komplexita: S rostoucí složitostí projektu se zvyšuje počet bloků nebo prvků ve vizuálním rozhraní, což může vést k přeplnění obrazovky. To může ztížit orientaci a navigaci ve vizuálním prostředí, což ztěžuje pochopení celkové struktury projektu. (Verma, 2022)

5.3. Užívání v praxi

Dnes je hojně používáno jak k programování her, tak i k dalším věcem jako je skriptování shaderů v Blenderu, který je zmíněn níže.

5.3.1. Scratch

Je světoznámý programovací jazyk vytvořený roku 2005. Je i dnes hojně užíván pro vzdělávací účely. Podle stránek Scratch je největším světovým společenstvím programujících pro děti a programovacím jazykem s jednoduchým vizuálním rozhraním, které umožňuje mládeži vytvářet digitální příběh, hry a animaci. (Scratch)
Jedná se o internetovou aplikaci, která obsahuje knihovnu již vytvořených her uživateli, tak tvoření her samotných a velice obsáhlou dokumentaci, podle které se naučí
i začátečník.



Obrázek 5 Scratch –vizuální programování (zdroj: vlastní)

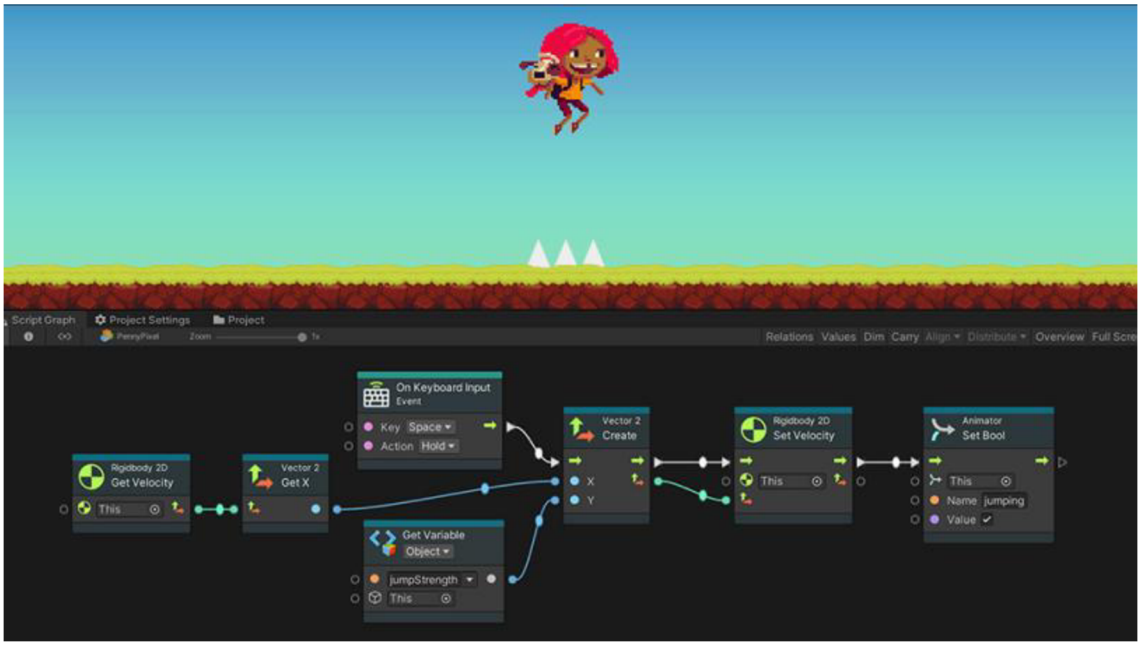
V příkladu výše, je vidět, že po kliknutí na zelenou vlajku se provede cyklus o 4 opakováních, ve kterých panáček posune o deset jednotek, následně otočí o 90 stupňů a počká půl vteřiny. Toto panáčka ze začátku dostane opět do stejného stavu po 2 vteřinách.

Scratch umožňuje vytváření hodně komplexních her, které jsou k vidění na stránce:
HTTPS://SCRATCH.MIT.EDU/EXPLORE/PROJECTS/ALL

5.3.2. Unity

Z počátku samotný Unity engine neměl funkci vizuálního programování implementovanou, nicméně v roce 2021 nastala změna. Dle dokumentace byla tato funkce přidána do verze 1.5.1-pre.3, dnes je ve verzi 1.8. (Unity)

Samotný proces programování je velice podobný Unreal Engine, který je zmíněn dále.



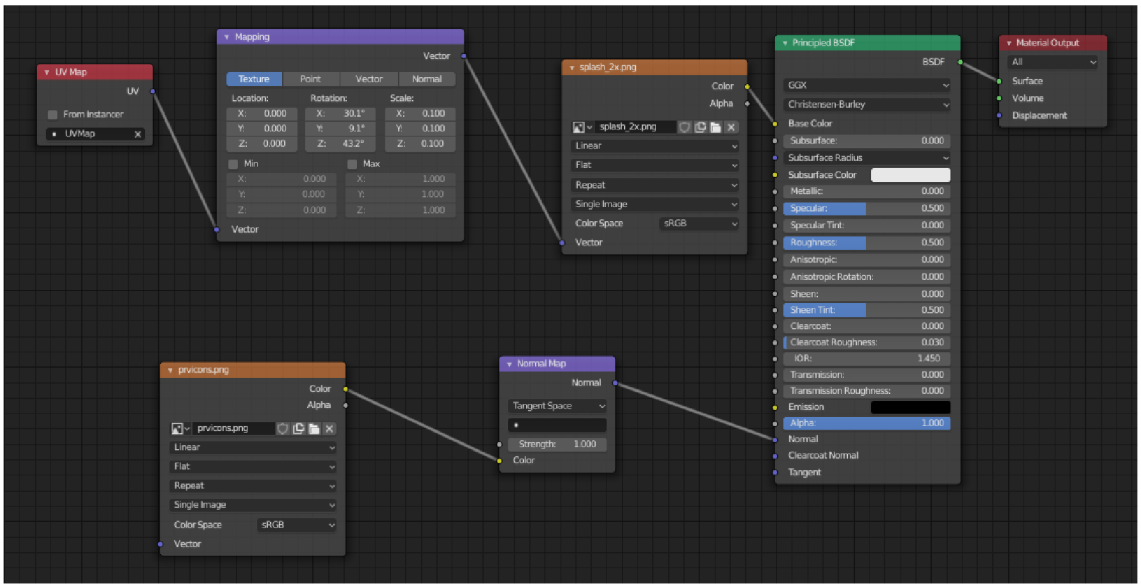
Obrázek 6 Unity – vizuální programování (zdroj: unity.com)

Z obrázku je zřejmé, že po kliknutí mezerníku bude k momentální rychlosti panáčka přičtena rychlost výskoku na ose Y o síle, která vrací funkce *Get Variable*, to zapříčiní skok panáčka se stejnou rychlostí na ose X. Ovšem pouze budeme-li usuzovat, že funkce *Get Variable* vrací číslo, které reprezentuje sílu skoku.

5.3.3. Blender

Vizuální programování nemusí být využito čistě k programování. V programu Blender se používá k tvoření tzv. shaderů, které definují vzhled materiálu, který je na povrchu nějakého 3D objektu nebo jeho části.

Shader je další část kódu, která se přidává do vykreslování scény (nebo jiné postprodukční metody). Shadery mohou měnit vizuální efekty tím, že mění způsob zobrazení textur a osvětlení, někdy způsobem, který by byl jinými technikami nemožný.
Je vyvinut tak, aby využíval grafický procesor počítače a odlehčil procesoru pro jiné úlohy (například výpočet fyzikálních interakcí). Předává informace o vrcholech a texturách grafickému procesoru, který pak vrací zpět informace o osvětlení a barvách. (Denham)



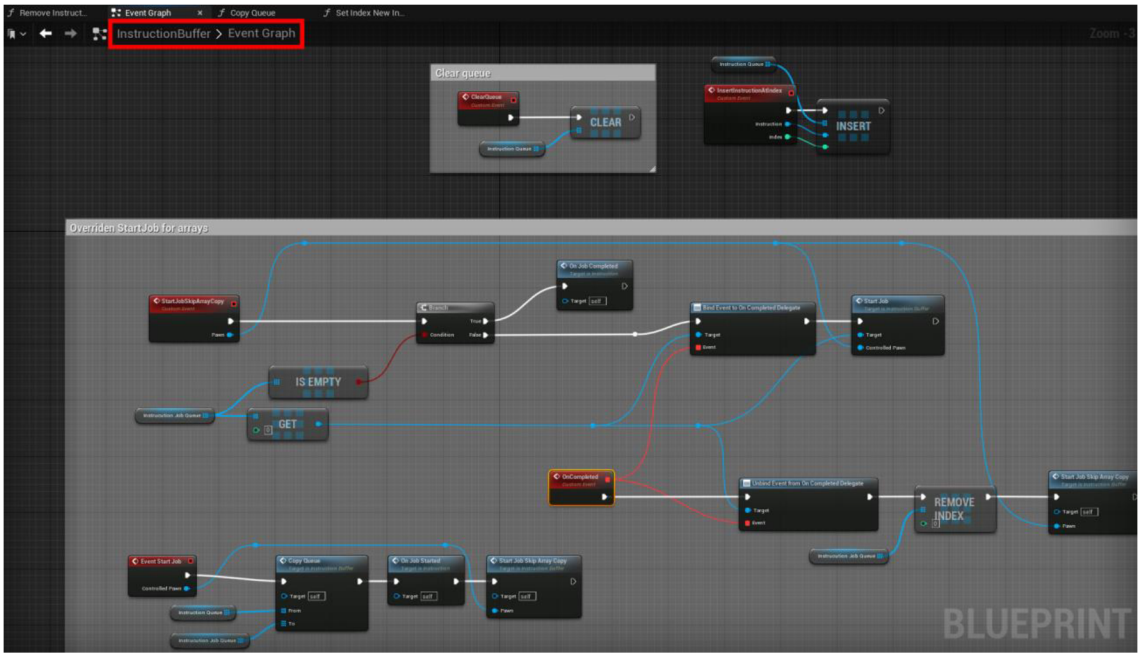
Obrázek 7 Blender – shader graph (zdroj: blender.org)

Na obrázku lze spatřit, že používá bloky, které jsou spojeny určitými čárami. Čáry fungují jako relace mezi samotnými parametry a jejich hodnotami. Samotné zpracování grafu probíhá z levé strany do pravé a končí výsledným blokem *Material Output*, kde je definovaná celá potřebná struktura materiálu.

5.4. Užití vizuálního programování v Unreal Engine

Unreal Engine poskytuje několik vizuálních programovacích nástrojů, které umožňují tvůrcům a vývojářům vytvářet hry a interaktivní vizuální obsah bez nutnosti psaní kódu. Zde je několik příkladů využití v tomto programu:

1. Blueprint: Blueprinty jsou vizuální programovací nástroje, které umožňují tvůrcům her vytvářet hratelnost, logiku a interakci mezi herními objekty pomocí grafického rozhraní. S blueprinty lze vytvářet kompletní herní systémy, jako jsou umělá inteligence postav, herní události, animace, fyzika a další.
2. Material Editor: Umožňuje tvůrcům her vytvářet a upravovat materiály pro povrchy a textury herních objektů. Materiály definují vzhled a vlastnosti povrchů, jako jsou barvy, stíny, lesk, transparentnost a další. Material Editor umožňuje tvorbu materiálů pomocí vizuálního rozhraní, které zahrnuje grafické uzly, které se spojují do sítě a definují chování materiálů.
3. Particle System: Umožňuje tvůrcům her vytvářet a animovat částice, jako jsou ohně, dým, prach, exploze a další. Particle System umožňuje tvorbu složitých vizuálních efektů pomocí vizuálního rozhraní, které zahrnuje grafické uzly a parametry, které ovlivňují chování částic.



Obrázek 8 Unreal Engine 5 – Blueprinty (zdroj: vlastní)

6. Využití her ve vzdělání

Edukativní počítačové hry jsou herní aplikace, které jsou navrženy takovým způsobem, aby hráčům pomáhaly naučit se nové věci, rozvíjet své schopnosti a znalosti, nebo se učit řešit problémy. Tyto hry jsou často používány v školách a vzdělávacích institucích jako nástroj pro podporu výuky, ale mohou být rovněž používány pro osobní a profesní rozvoj. Edukativní počítačové hry mohou mít pozitivní vliv na rozvoj člověka v mnoha ohledech. Například mohou pomoci hráčům lépe porozumět určitým tématům nebo principům, rozvíjet dovednosti jako kritické myšlení, logické uvažování, řešení problémů a komunikace, a zlepšit jejich schopnost učit se nové věci. Kromě toho mohou edukativní počítačové hry být zábavným a motivačním způsobem, jak se učit a získávat nové znalosti.

6.1. Přínos a rozvoj pro uživatele

Edukativní počítačové hry mohou přinášet různá pozitiva, záleží na tom, jakým způsobem jsou navrženy a jakým způsobem jsou hráči používány.

6.1.1. Logické myšlení a paměť

Užití logického myšlení je nedílnou součástí hraní jakékoliv hry, tj. když hra není zaměřená na logické myšlení, je nutné určitým způsobem přemýšlet. Taková hra může být zaměřena na matematické nebo jiné logické myšlení. Spoustu těchto her lze využívat už u předškolních dětí, kdy se děti učí jednoduchému počítání, čtení či plnění dalších úkolů.

Paměťové hry většinou vyžadují pozastavení nad problematikou, koncentraci, myšlení a zapamatování si určité informace, která je vyžadována k dalšímu pokračování.

6.1.2. Postřeh

Mezi tyto typy her se většinou řadí střílecí hry, kdy hráč musí vnímat jak dění v momentální situaci, okolí tak i své nepřátele. Není pochyb, že tyto hry nebudou doporučovány jako vzdělávací, nicméně to je dobrý příklad.

6.1.3. Kreativita

Nejčastější žánr právě těchto her je tzv. sandbox, kdy hráč má za úkol něco stavět, tvořit nebo navrhovat. Většinou hra nemá příběh a dává uživateli volnou ruku nad obsahem, který vytvoří. Např. hra Minecraft (2009), která má kompletně náhodnou stavbu světa (tzv. generování světa) dává možnost hráči kompletně přetvořit každý kousek herního světa, které jsou v mezích hranic samotné hry.

6.1.4. Koncentrace

„Každý hráč počítačových her ví, o kolik lépe se vyhrává, je-li člověk na problém hry soustředěný. Stejné pravidlo platí i v reálném životě, takže počítačové hry jsou v tomto smyslu "školou života".“ (Janovský)

6.1.5. Motorika

„Hráči her, v nichž záleží na rychlosti, si zdokonalují postřeh, nervosvalovou souhru a motoriku. Hry hrané myší pomáhají nacvičit pohyb ukazatelem, který se hodí i v užitkových programech.“ (Janovský)

6.1.6. Jazykové znalosti

Naprostá většina her je v angličtině, a proto je uživatel z části nucen si rozvíjet slovní zásobu, popřípadě výslovnost. V Česku se mnoho studentů naučí právě nejvíce anglických slov z počítačových her. (Krejčí)

6.2. Negativní stránka

Může se zdát, že v dnešní době učení moderními technologiemi má samé výhody, nicméně to není úplně pravda. Učení hrou je určitě skvělé a efektivní, nicméně pokud se nedodržují určitá pravidla, jakožto styl sezení, může se projevit i negativní stránka učení tímto způsobem.

6.2.1. Omezený rozpočet

Je jedním z problémů, nicméně záleží na požadavcích uživatele, nový středně výkonný počítač může stát až 20 tisíc Kč, naopak bazarový může být mnohem levnější. Mnohem více uživatelů používá další zařízení, jako jsou mobily, které jsou mnohdy levnější než již zmíněný počítač.

Stránka BroadbandSearch uvádí, že do ledna roku 2022 byla mobilní zařízení využívána pro hledání na internetu o 5 % více vůči osobním počítačům. Mobilní telefony tedy cca 55 %, osobní počítače až 42 % a zbytek jsou tedy další zařízení. (BroadbandSearch) V důsledku vyššího využívání mobilních zařízení k hledání obsahu na internetu lze předpokládat celkově vyšší využívání těchto zařízení.

6.2.2. Ovlivnění psychiky

Násilí a agrese může být dalším ovlivňujícím faktorem hraní her.
„Za jedno z největších rizik v souvislosti s počítačovými hrami je považováno ztotožnění se s postavou, za kterou v dané hře hrajeme. Podle odborníků jsou nejpočetnější rizikovou skupinou děti do věku 12 let. Hry člověku předávají velké množství informací, které dítě nemusí umět zpracovat, bude na ně nekriticky nahlížet nebo se dokonce může činy, které se před ním odehrávají, inspirovat.“ (Fuka)

6.2.3. Fyzické problémy

Z důvodu absence pohybu je zde zvýšená šance obezity. Špatným stylem sezení mohou nastat svalové problémy anebo problémy s kosterní soustavou.

6.3. Užití her k seberozvoji

„Právě možnost rozvoje osobnosti nabízí využití počítačové hry ke vzdělávacím účelům. Proces hraní hry v člověku vyvolává soustředěnost celkovou, ale i soustředěnost zaměřenou na zisk, výhru, cíl. Obdobný mechanismus můžeme sledovat i v procesu učení — pokud se chceme něco naučit, získat informace, pak naši pozornost soustředíme na práci a naším cílem je potřebné informace získat, porozumět jim a interpretovat je, podobně jako v Bloomově taxonomii vzdělávacích cílů.“ (Kašpar, 2020)

„Považujeme-li svět počítačových her za simulaci reálného světa, pak s sebou přináší i možnost ocitnout se v situacích, které by buď v realitě nastat nemohly, nebo je velice obtížné je v reálném světě simulovat, a právě v těchto situacích vyzkoušet naše jednání, instinkty, znalosti. Do této kategorie můžeme zařadit například simulátory operací v lékařském prostředí nebo trenažery v autoškolách, i když se nejedná o počítačovou hru v pravém smyslu slova — řekněme, že jsou to „vzdělávací počítačové simulace.“ (Kašpar, 2020)

7. Analýza a návrh aplikace

Aplikace se bude inspirovat hrou Karel, který využívá vlastní stejnojmenný programovací jazyk. V této hře má hráč šachovnici, na které je umístěný robot pojmenovaný Karel. Karel umí nějaké základní příkazy jako jsou *krok vpřed*, *otoč se vlevo* apod. S pomocí těchto základních funkcí je hráč schopen dávat instrukce robotu, který je následně posloupně zpracovává.

(Kadlec) uvádí na své osobní stránce uvedený kód níže. Takto vypadala původní syntaxe jazyku.

```
BEGINNING-OF-PROGRAM


 DEFINE turnright AS
 BEGIN
   turnleft
   turnleft
   turnleft
 END


 BEGINNING-OF-EXECUTION
   ITERATE 3 TIMES
     turnright
```

Uživatel je schopen tvořit sekvence, smyčky nebo podmíněné příkazy.

Ve tvořené aplikaci bude princip podobný, nicméně uživateli nebude dávat možnost psaní kódu a vše se bude řešit přes vizuální bloky.

Uživatel bude též schopen tvořit vlastní funkce, které je bude možné volat, a to i rekurzivně. Postava, kterou bude hráč ovládat bude též umístěna do šachovnice, akorát ve 3D prostředí. Ve hře bude možné projít různými úrovněmi, které uživatele budou nutit nad zamyšlením určité situace, přičemž náročnost jednotlivých úrovní by se měla zvyšovat postupným procházením. Bude zde i možnost volného módu, kdy uživatel nebude mít žádný úkol a může si tvořit co chce.

7.1. Výběr technologií a nástrojů pro vývoj

Jak již bylo zmíněno v předchozích kapitolách, pro vývoj samotné hry se bude používat Unreal Engine ve verzi 5.1. Především bylo tak zvoleno z důvodu jednoduchého pochopení bez nutnosti psaní řádků kódu. V rámci této práce bylo představeno vizuální programování, které se též v Unreal Engine hojně využívá.

K tvorbě 3D modelů se bude používat Blender, což je velice schopná open-source modelovací aplikace. Model se vymodeluje, nastaví se mu veškeré potřebné parametry a následně exportuje do FBX formátu. Soubory tohoto formátu lze následně importovat do Unreal Enginu a dále s ním pracovat.

Pro přívětivost uživatelského rozhraní se použije online aplikace Figma. V ní lze plánovat UX, UI, a dokonce hromadně online spolupracovat na stejném projektu v reálném čase. Především se využívá pro návrhy designu pro webové aplikace, ale nemusí tomu tak být.

„User Experience (UX) neboli uživatelská zkušenost se netýká pouze webdesignu. UX design se zabývá navrhováním různých řešení tak, aby co nejlépe sloužila uživateli.” (Leška, 2020)

8. Implementace aplikace

Jako první je důležité si rozmyslet, čím přesně začít. V případě, že aplikace je do detailu vymyšlená, pak lze začít již tvořením herních souborů jako jsou modely, zvuky, obrázky apod. V opačném případě je doporučeno nejdříve provést návrh a následně pokračovat již zmíněným tvořením herních souborů.

8.1. Získání nebo vytvoření herních assetů

Z počátku vývoje hry se typicky prototypují herní 3D modely. Finální modely se tvoří až ve chvíli, kdy je určen styl, či směr vývoje. Tedy na začátku vývoje vypadala herní postava jako krychle a herní svět reprezentovala levitující platforma, kde se veškeré dění odehrávalo. Když už byla dokončena programová část, tak se tyto modely nahrazovaly novými, které splňovaly stejný grafický styl.

8.2. Rozplánování implementace jednotlivých částí

Vývoj hry se dělí na několik částí. Jako první se řešila samotná logika zpracování instrukcí postavy, s tím se pojilo i postupné vytváření uživatelského rozhraní a v závěru modelování 3D objektů.

- **Rozplánování UI a UX**

  Uživatelské rozhraní neboli UI, je navrženo co nejintuitivnějším způsobem.

  V hlavní nabídce lze najít rozcestník, kde si lze vybrat mezi herním módem *Úrovně*, *Sandbox*, možnostmi hry a vypnutím. Po kliknutí na tlačítko *Úrovně* se uživateli zobrazí další nabídka, kde si lze vybrat jakoukoliv úroveň, každá úroveň má název a krátký popis situace, kterou bude muset uživatel řešit.

  V případě kliknutí na *Sandbox* bude uživatel přesunut na nastavení mapy, kde si zvolí velikost šachovnice a následně je přesunut na samotnou mapu.

  V možnostech lze zobrazit tvůrce hry a assetů. Nastavení grafických detailů zde není, protože hra je tvořena v low poly grafické stylizaci a není potřeba měnit jejich nastavení, protože by nedošlo k výrazné vizuální změně a nárůstu výkonu.

Low poly model je typ 3D modelu, který se vyznačuje minimálním počtem polygonů (trojúhelníků). Tyto modely jsou charakteristické pro jednoduchou geometrii, s minimálním množstvím detailů a výraznými hranami. Low poly modely jsou často používány v počítačových hrách, simulacích a vizualizacích, kde je důležité dosáhnout co největšího výkonu při co nejnižší náročnosti na hardware. (Fuentes, 2021)

- **Rozplánování ovládání**

Hra se bude ovládat pomocí myši a klávesnice. Herní postava bude plnit instrukce pomocí vytvořené hierarchie v uživatelském rozhraní, jednotlivé instrukce půjdou přetahovat z pravé strany ze záložky dostupných instrukcí nebo vlastních funkcí do levé, kde se bude nacházet samotný prostor pro skládání instrukce. Bude zde tlačítko pro zahájení spouštění instrukcí, obnovení postavy na výchozí pozici a zastavení momentálně vykonávané instrukce.

8.3. Implementace jednotlivých částí

Jak již bylo zmíněno, práce se nejdříve zaměřuje na programovou část a až následně na tvoření herních souborů jako jsou 3D modely a textury.

Programová část je hlavním předmětem vytvořené aplikace a této práce. Na grafický vzhled nebyl brán takový ohled a čas strávený na této části byl mnohem kratší.

Postupné vytváření herních funkcionalit je popsané níže v jednotlivých odstavcích.

Vsuvka: Typicky se v programování používá *camelCase* nebo *snake_case* zápis. Unreal Engine je kompletně napsaný v *PascalCase*, a tak jsou formulovány i ukázky. Jedná se o oficiální standard přímo z oficiálních *guidelines*, které jsou dostupné v dokumentaci.

Stručné shrnutí následujících kapitol. Nejprve se zaměřují na samotnou logiku zpracovávání instrukcí, následně na uživatelské rozhraní a funkcionality spojené s ním.

Další část je věnována hernímu světu, ve kterém se postava nachází a pohybuje.

Všechny následující kapitoly jsou věnovány nezbytně nutným funkcionalitám, bez kterých by hra nemohla fungovat.

Uživatelskému rozhraní jako hlavní nabídka, výběr úrovní apod. není v práci věnován čas, nicméně nejedná se o nijak výjimečnou funkcionalitu, která by se lišila od ostatních her. Více informací ohledně konkrétní implementace lze najít na straně č. 52.

- **Třída Instruction**

Nejprve bylo potřeba vytvořit třídu, která bude zpracovávat libovolnou aktivitu a bude dostatečně abstraktní pro tvorbu komplexnějších událostí. Třída dědí přímo z *Object* třídy, tzn. objekt na bázi této třídy existuje pouze v paměti a není umístěn nikde v herním světě.

Nějak takto by mohl vypadat UML class diagram:

Instruction
+ OnCompleted: Delegate - InProgress: bool - TakesTime: bool + MenuDisplayClass: Class<Widget> + DragInstructionDisplayClass: Class<Widget> + InstructionDisplayClass: Class<Widget> + StartJob(Pawn* ControlledPawn, InstructionExecutionContext* Ctx): void + OnJobCompleted(InstructionException* Exception, InstructionExecutionContext* Ctx, Pawn* ControlledPawn): void + CompleteJob(InstructionException* Exception, InstructionExecutionContext* Ctx, Pawn* ControlledPawn): void + IsInProgress(): bool

UML Class Diagram 1 Popis instrukční třídy (zdroj: vlastní)

Je zde patrné, že instrukce jsou nezávislé od postavy, jejž chování je kontrolováno.

Pro správné vykreslování v hierarchii má každá instrukce informace o tom, jaké bloky má přiřazené v uživatelském rozhraní. Každý vytvořený widget má přístup k *Instruction* atributu, kam je následně instrukce přiřazena. Tato funkcionalita slouží primárně ke znovu použití stejného widgetu pro více typů instrukcí.
Jedná se o tyto atributy:

- MenuDisplayClass – Třída widgetu, která je zobrazena ve výběru instrukcí
- DragInstructionDisplayClass – Třída widgetu, která je zobrazována v době vybrání instrukce a následném tažením myši. Často je shodná s MenuDisplayClass.
- InstructionDisplayClass – Třída widgetu, která je zobrazována v hierarchii instrukcí.

U každé třídy lze zjistit stav, jestli probíhá, či nikoliv pomocí *IsInProgress* metody. Lze též naslouchat na ukončení dané instrukce, jedná se o atribut *OnCompleted*, který je typu *Delegate*.

Delegate v Unreal Enginu je speciální typ funkce, která umožňuje předávat volání funkcí mezi třídami a komponentami v rámci hry. Je definován jako proměnná, která uchovává referenci na funkci a může být předána jiné třídě nebo komponentě jako argument. (Unreal Engine)

Obecné vykonávání instrukce funguje následovně:

1. Volá se metoda *StartJob* na instrukci, předává se instance třídy InstructionExecutionContext a kontrolovaná postava (parametr ControlledPawn).
2. V implementaci metody *StartJob* se volá rodičova stejnojmenná metoda, ve které se nastaví *IsInProgress* atribut na hodnotu *true*. Bez volání rodiče by nebylo zřejmé, zda je instrukce v běhu. Následně vykonává vlastní implementace. Více na straně o implementaci instrukce pohybu.
3. Ve chvíli, kdy je proces instrukce dokončen se volá metoda *CompleteJob*, kde se předávají informace o tom, zda byla úspěšně dokončena a *InstructionExecutionContext* objekt.

InstructionExecutionContext je rozebrán dále v práci na straně č. 33.

- **Třída InstructionBuffer a sekvenční zpracovávání instrukcí**

Třída *InstructionBuffer* je určena pro sekvenční zpracování instrukcí, zatímco je sama třídou instrukce. Z této třídy tedy dědí.

Objekt na bázi této třídy obsahuje pole dalších instrukcí, které má za úkol zpracovat. Pole instrukcí, které má zpracovávat je ve zdroji hry pojmenováno jako InstructionQueue, tedy fronta, čekající na zpracování.

Implementace je složitější než v předchozím případě, ve stručnosti dochází k:

1. Zkopíruje se pole *InstructionQueue* do pole *InstructionJobQueue*

    1.1. Kopírování se děje, protože tento objekt existuje po celou dobu zapnuté hry. V případě, že by byla instrukce zavolána, by bylo pole *InstructionQueue* prázdné, protože v předchozím případě došlo k celkovému nebo částečnému vyprázdnění.

2. Volá se metoda *StartJobSkipArrayCopy* s instrukcí na první pozici v poli InstructionJobQueue, ve které je implementace následovná

    2.1. Zjistí se z kontextu instrukce, zda má být ukončena, pokud ano, tak se ukončí pomocí *CompleteJob* metody.

    2.2. Kontroluje se, zda je pole InstructionJobQueue prázdné, pokud ano, tak je instrukce u konce a volá se *CompleteJob* metoda.

    2.3. Zjišťuje se, zda je instrukce zaseklá v nekonečné smyčce, pokud ano, tak instrukce končí s výjimkou *InstructionCallStack.* (více v kapitole o výjimkách v instrukcích)

    2.4. Začne se naslouchat na ukončení instrukce z parametru. (*OnCompleted* atribut)

    2.5. Volá se *StartJob* metoda na instrukci z parametru a předávají se jí potřebné parametry.

    2.6. Ve chvíli, kdy je instrukce dokončena, tak se děje následovné:

        2.6.1. Přestane se naslouchat na ukončení instrukce.

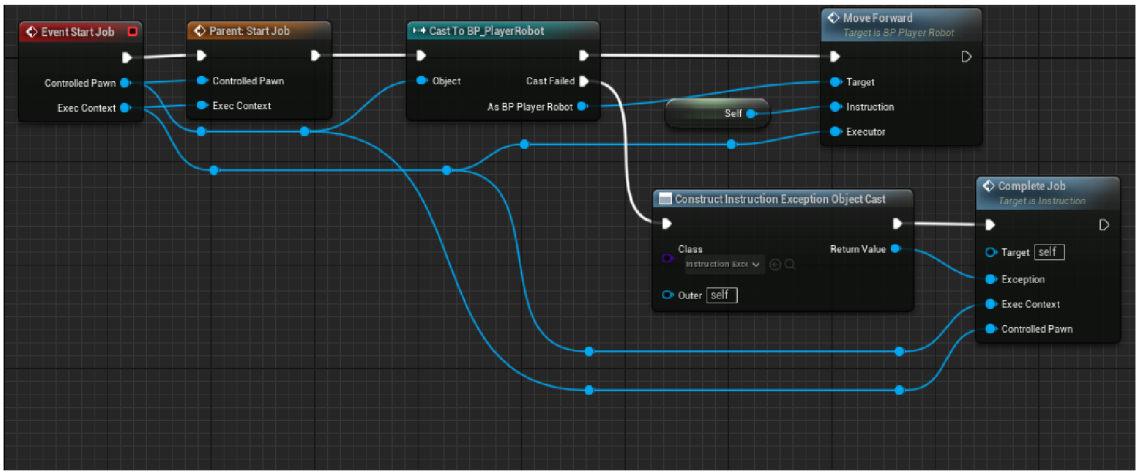
        2.6.2. V *InstructionJobQueue* se odebere se prvek na nulté pozici.

2.6.3. Spouští se metoda *StartJobSkipArrayCopy* a cyklus běží znovu od 2. bodu.

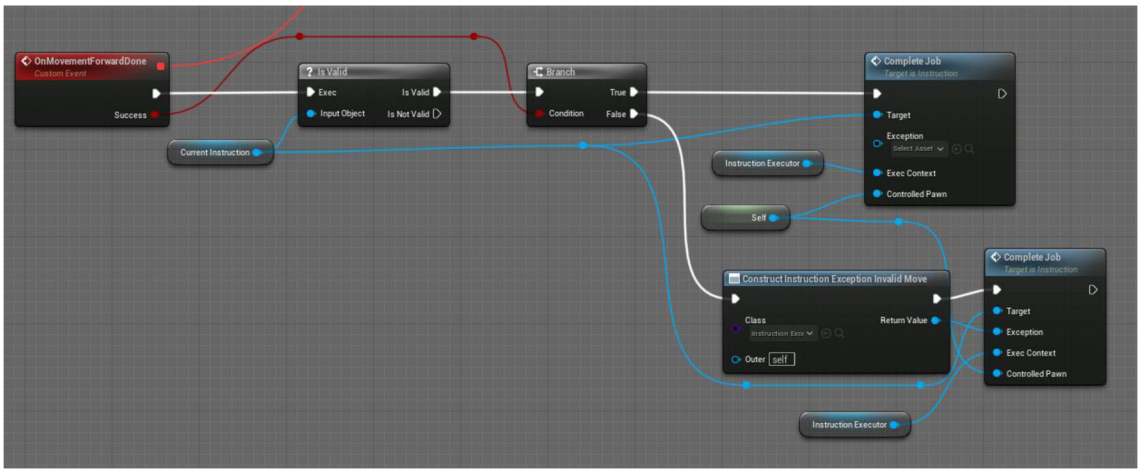
Z této metody dědí například kondicionální a cyklující instrukce.

- **Implementace instrukce pohybu**

Tato kapitola o instrukci je zde vedena víceméně jako demonstrace implementace instrukce.



Obrázek 9 Implementace MoveForward instrukce (zdroj: vlastní)



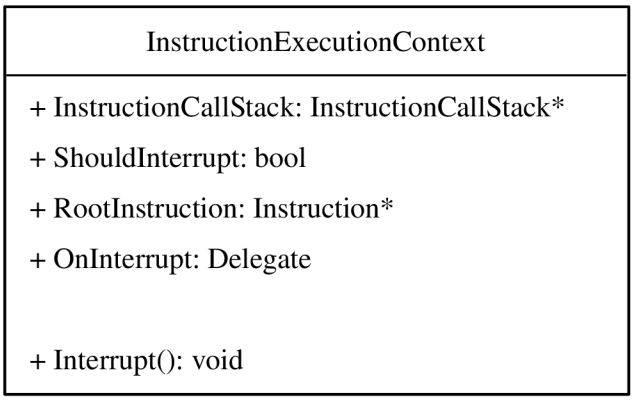
Obrázek 10 Implementace MoveForward instrukce ve třídě postavy (zdroj: vlastní)

Na obrázku č. 9 je vidět volání metody *MoveForward* s parametrem instrukce na přetypovaném objektu kontrolované postavy (*BP_PlayerRobot*). Na spodním obrázku č. 10 je vyobrazeno dokončení instrukce, které nastává ve chvíli, kdy je postava na finálním místě. Je volána metoda *CompleteJob* na předané instrukci přes parametr instrukce. Tímto je instrukce u konce.

- **Instrukční kontext**

Pro udržení informací v průběhu zpracovávání jednotlivých instrukcí je použita třída *InstructionExecutionContext*. Tento objekt je vždy tvořený před začátkem spuštění kořenové instrukce a je postupně předáván mezi nimi dále.

UML class diagram vypadá následovně:



UML Class Diagram 2 Popis instrukčního kontextu (zdroj: vlastní)

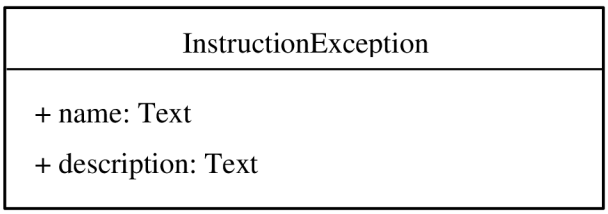
Instrukční kontext osahuje informaci o tom, zda bylo požádáno o zastavení spouštění dalších instrukcí. O samotné zastavení se stará metoda *Interrupt*, která nastaví atribut *ShouldInterrupt* na hodnotu *true*, následně volá *OnInterrupt* delegate a *CompleteJob* metodu na *RootInstruction* atributu.

Též je zde i objekt *InstructionCallStack*, který je zmíněn dále v práci na straně č. 34.

- **Výjimky v instrukcích**

V herním poli se mohou stát věci, které je třeba ošetřit. Například postava se nemůže dostat ven z herního pole, poté je vrácena výjimka.

Výjimka je třída na bázi třídy Object a její diagram by mohl vypadat následovně:



UML Class Diagram 3 Popis třídy pro výjimky v instrukcích (zdroj: vlastní)

Jedná se objekt, který obsahuje pouze název a popis.

Výjimka se vždy kontroluje při dokončení instrukce v *InstructionBuffer* objektu. Kontroluje se její platnost pomocí nulové, či nenulové hodnoty.
Objekt této třídy se vrací v *CompleteJob* metodě, jak je tomu na obrázku č. 9 v případě neplatného přetypování. Jestliže na výjimku dojde, tak je v uživatelském rozhraní zobrazeno dialogové okno s chybovou hláškou a je přerušeno další zpracovávání.

- **Instruction Call stack**

Call stack je v informatice označení pro zásobník, který ukládá informace o aktuálně probíhajících funkcích nebo metodách v programu. Každá nově volaná funkce se přidá na vrchol zásobníku a po dokončení se opět z něj odstraní. Call stack umožňuje programu udržovat správnou sekvenci volání funkcí a návratových hodnot. (MDN)

Samotný engine má protekci před rekurzivním call stackem. Děje se to ve chvíli, kdy je metoda nebo jakákoliv funkce volána rekurzivně, typicky volá sama sebe. Když tento proces ve hře nastane, je nutné její ukončení. Aby hra ukončení předešla, tak je zde proces detekce této situace v rámci zpracování instrukcí. V případě, že tato situace nastane, tak je vrácena výjimka typu *InstructionCallStack*. Toto může nastat pouze ve chvíli, kdy je instrukce špatně implementována.

Proces řešení se děje pouze v objektech na bázi *InstructionBuffer* a je následovný:

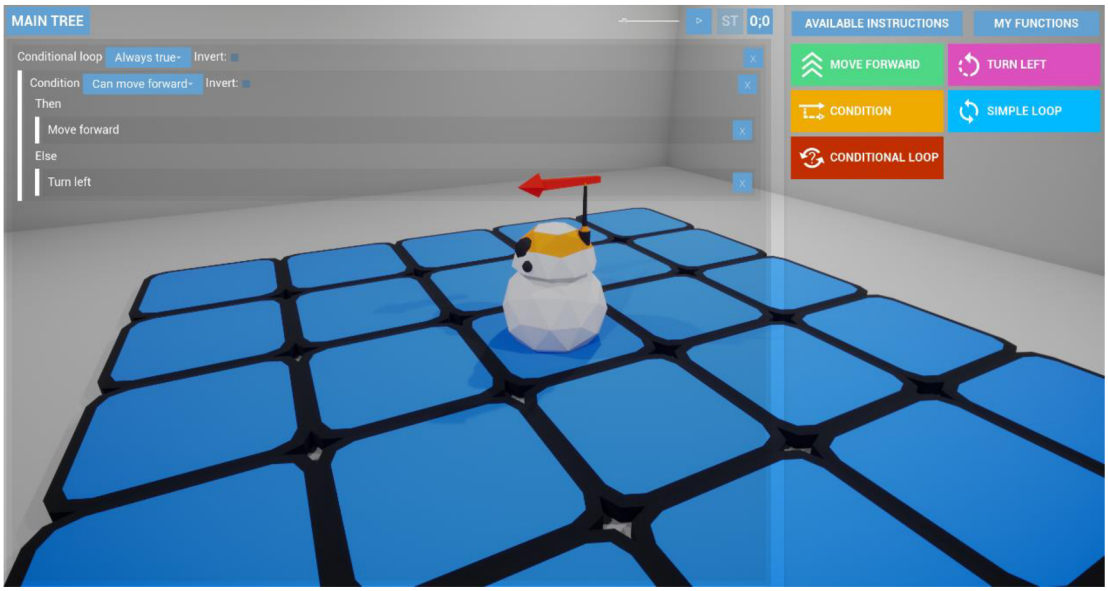
1. V metodě *StartJobSkipArrayCopy* se kontroluje, zda instrukce existuje v zásobníku instrukcí.
    1.1. Pokud ano, tak se kontroluje, zda vyžaduje čas pro zpracování. (atribut *TakesTime*)
        1.1.1 Pokud vyžaduje čas pro zpracování, tak nemůže být instrukce v nekonečné smyčce. Alespoň ne pro detekci v enginu a následovného vypnutí aplikace.
    1.2. V opačném případě instrukce končí s výjimkou zaseknutí.
2. Před voláním *StartJob* instrukce se přidá do zásobníku.
3. Ve chvíli, kdy je instrukce dokončena (naslouchání na *OnCompleted* delegate) a instrukce vyžaduje čas na zpracování, tak je zásobník vyprázdněn.

Tento proces si neuchovává všechny instrukce, ale pouze ty, které nečerpají minimální čas procesoru jako je například nekonečný cyklus.

Situace, kdy by instrukce mohla skončit v tomto stavu je, kdyby se volala instrukce cyklu s kladnou hodnotou, protože *InstructionBuffer* se automaticky ukončuje v případě prázdné fronty instrukcí. Toto by způsobilo nekonečnou smyčku, která by maximálně vytěžovala procesor, a proto by byl proces aplikace po detekci této situace ukončen. Úkolem této třídy je tedy zachycení smyčky ještě před detekcí v enginu samotném.

- **UI instrukcí**

UI tvoření instrukcí závisí na několika aspektech. Počet dostupných instrukcí je závislý na herních úrovních, kdy v prvních jsou omezené pouze na potřebné, aby uživatel nebyl zmatený a podrobně si prozkoumal jejich funkcionalitu. Vytváření vlastních funkcí je závislé na stejných aspektech.



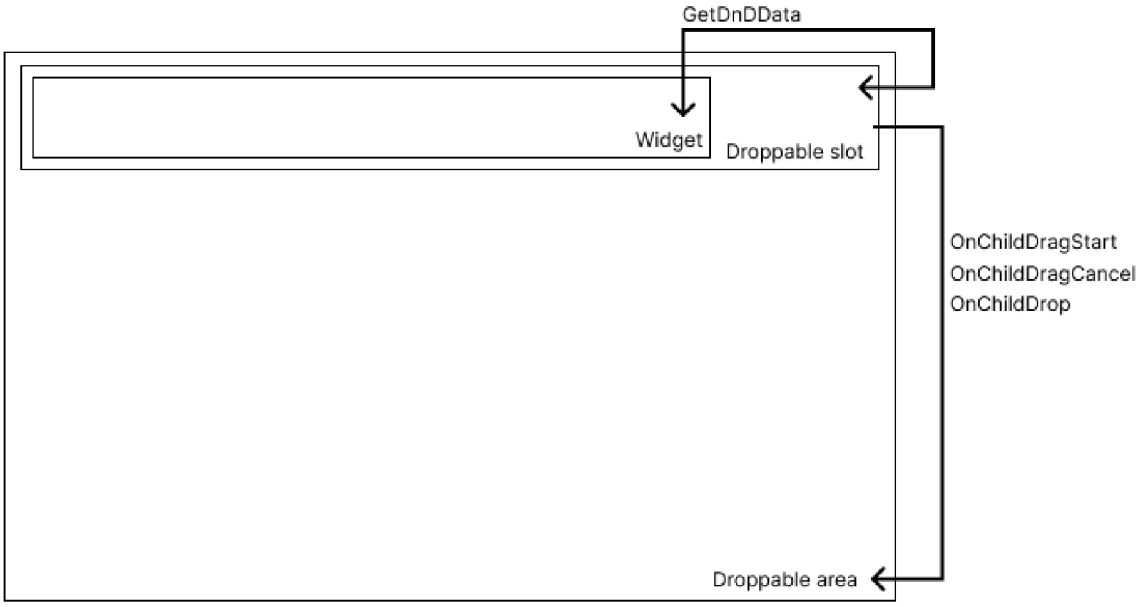
Obrázek 11 UI – Hierarchie instrukcí (zdroj: vlastní)

Jak je z obrázku patrné, na levé části se nachází hierarchie instrukcí, kterou tvoří uživatel tažením instrukčních bloků z pravé strany. Nad touto hierarchií se nachází záložky s hierarchiemi instrukcí, tj. při tvoření vlastních funkcí se zde implementuje jejich obsah. Na pravé straně jsou tlačítka na kontrolu postavy, lze zapnout hierarchii, zastavit chod, či restartovat pozici a rotaci na původní hodnoty.

Na pravé straně se nachází množina instrukčních bloků, které lze v úrovni použít.
V případě možnosti tvoření vlastních funkcí se zde objeví záložka *My functions,* ve které lze přidávat vlastní funkce, měnit jejich implementaci, či název.

- **Mechanika drag and drop instrukcí**

Nejdříve se navrhovala a tvořila obecná mechanika, nezávislá na funkci instrukcí.



Obrázek 12 Drag and Drop – komunikace mezi widgety (zdroj: vlastní)

Obrázek výše popisuje stručnou funkcionalitu této mechaniky, která bude dále podrobněji rozebrána.

- **DroppableArea**

Jako první je nutno implementovat widget *DroppableArea*, který zpracovává pořadí potomků, jejich výměnu pořadí, odstraňování nebo přidávání. Na tyto události lze naslouchat pomocí několika atributů typu *Delegate*.

Konkrétně se jedná o atributy:

- OnChildDragStart – Volá se ve chvíli, kdy je potomek zvednut a tah myší začal.
- OnChildDragCancel – Volá se ve chvíli, kdy je tažení přerušeno.
- OnChildDrop – Volá se ve chvíli, kdy je potomek upuštěn.

- **DroppableArea a přidávání potomků**

Aby mohly být widgety jakéhokoliv typu, je nutné si držet jejich pole (v projektu nazýváno *Children*). Při překreslování se vezme každý potomek, obalí se neviditelným widgetem *DroppableSlot*, který v sobě uchovává informaci o pořadí a nadřazeném *DroppableArea* widgetu.

- **DroppableArea widget a změna pořadí, či odebírání potomků**

Pro změnu stačí změnit pořadí v poli potomků a překreslit je. Při odebírání platí to samé, akorát se z pole prvek odstraní a následně překreslí.

- **DroppableSlot widget a On drag mechanika**

Jakýkoliv widget obalen tímto widgetem je schopen být tažen myší.
Toto samozřejmě závisí na podřazeném widgetu, tedy tom, který je obalen, když implementuje určité rozhraní a přetěžuje metodu *GetDnDData*, která vrací *DragDropOperation* objekt. Ten je schopen tuto operaci tvořit.
DragDropOperation v Unreal Enginu je třída reprezentující operaci přetahování a upouštění v uživatelském rozhraní. Tato třída se používá pro přenášení dat mezi různými prvky uživatelského rozhraní, jako jsou například tlačítka, seznamy nebo okna. (Unreal Engine)

- **DroppableArea a On drop mechanika**

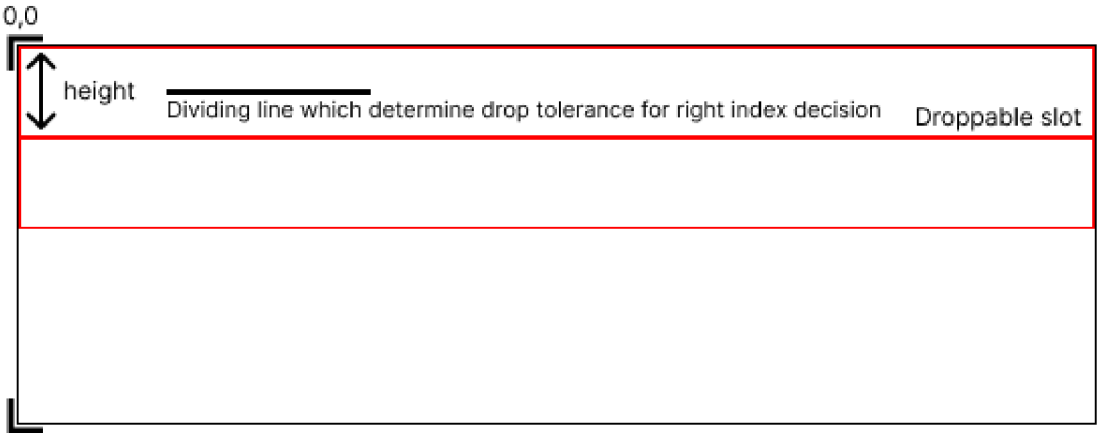
Ve chvíli, kdy je upouštěn widget do *DroppableWidget* oblasti, tak se děje následující:

1. Kontroluje se, zda oblast přijímá potomky. Jedná se o atribut *AcceptsChildren*.
2. Zkouší se přetypovat na vlastní třídu dědící *DragDropOperation*.
3. Kalkuluje se pozice, kam má být nový potomek vložen. Více v obrázku dole.
4. V případě, že je potomek již potomkem stejné oblasti, mění se jejich pořadí a zde proces končí.
5. Dále nutné kontrolovat, zda není potomek vkládán do nějakého jeho podřazeného potomka. Rekurzivně se tato situace kontroluje, pokud se tak děje, proces končí.

Kdyby se toto nekontrolovalo, bylo by možné, že se widget vloží do nějakého podřazeného potomka a následně při překreslení zmizí, protože jeho přípojný bod bude ztracen.

6. Pokud oblast pouze nekopíruje upuštěné widgety, je tažený widget odstraněn z předchozí oblasti.

Widget se vloží na vypočítanou pozici.



Obrázek 13 Drag and Drop – kalkulace pozice (zdroj: vlastní)

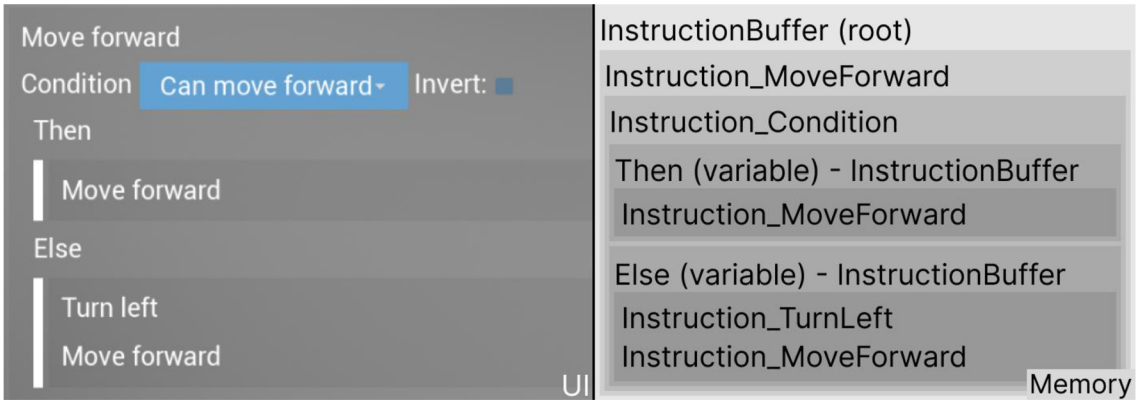
Pro vypočítání vhodné pozice je použito sčítání výšky existujících potomků.

Zjišťuje se pomocí relativní pozice myši k levému hornímu rohu. Je zde i tolerance poloviny výšky widgetu, aby se vkládaly jednodušeji, v případě, že by tomu tak nebylo, nebylo by možné vypočítat nultou pozici.

- **Propojení DroppableArea widgetu s instrukcemi**

K propojení těchto dvou widgetů je použit další widget (v kódu *InstructionFlow*), který propojuje *DroppableArea* widget a atribut *InstructionBuffer*.
V tomto widgetu se začne naslouchat na přidávání, odebírání, či změnu pozice prvku v *DroppableArea* widgetu.
Při přidání se zavolá metoda, pro kterou se naslouchá, z parametru se získá widget, ze kterého se získá instrukce a její nová pozice. Následně je tato instrukce přidána na danou pozici do atributu *InstructionBuffer*.
Při změně pořadí se z parametrů získá stará a nová pozice. V *InstructionQueue* v *InstructionBuffer* atributu je následně element ze staré pozice přesunut na novou.

Při odstranění pouze odstraní danou instrukci v *InstructionBuffer* na pozici z parametru. Celý proces funguje jako replikace stromu mezi *InstructionBuffer* a *DroppableArea* widgetem.



Obrázek 14 Propojení hierarchie instrukcí v UI s instrukcemi v paměti (zdroj: vlastní)

Na obrázku na levé straně je vidět hierarchie uživatelského rozhraní ve hře. Na pravé je vidět, jak jsou instrukce strukturovány v paměti. Kořen hierarchie je *InstructionBuffer*, do kterého jsou přidávány rekurzivně další a ty rekurzivně přidávají další atd. To stejné se děje v uživatelském rozhraní a tento strom je velice podobně strukturovaný.

- **Které instrukce zobrazit v UI?**

Pro držení informací o instrukcích apod., se používá instance třídy *GameDescriptor*, který je součástí objektu *GameInstance*.

Game instance v Unreal Engine je způsob, jakým se předávají hodnoty z mapy do jiné mapy. Jinými slovy, herní instance je trvalý objekt, který je vždy přítomen v každé mapě. (Bellincampi)

Následovně by mohl vypadat UML class diagram:

InstructionGameDescriptor
+ AvailableInstructions: Class<Instruction>[] + AvailableConditionals: Class<Conditional>[] + Pawn: Class<Pawn> + AllowCustomFunctions: bool

UML Class Diagram 4 Popis herního deskriptoru (zdroj: vlastní)

- AvailableInstructions – Dostupné instrukce
- AvailableConditionals – Dostupné podmínky (více na další straně)
- Pawn – Postava hráče
- AllowCustomFunctions – Dostupnost tvoření vlastních funkcí

V objektu jsou obsaženy instrukce, které lze vykonávat, herní postava a informace o tom, zda je povolené tvořit vlastní funkce.

Prvky v UI jsou automaticky přizpůsobeny právě tomuto objektu. Např. na obrázku č. 11 lze vidět vpravo nabídku instrukcí, která je vyplněna widgety, dostupných z každé instrukce v *AvailableInstructions* atributu. Konkrétně se jedná o *MenuDisplayClass* atribut.

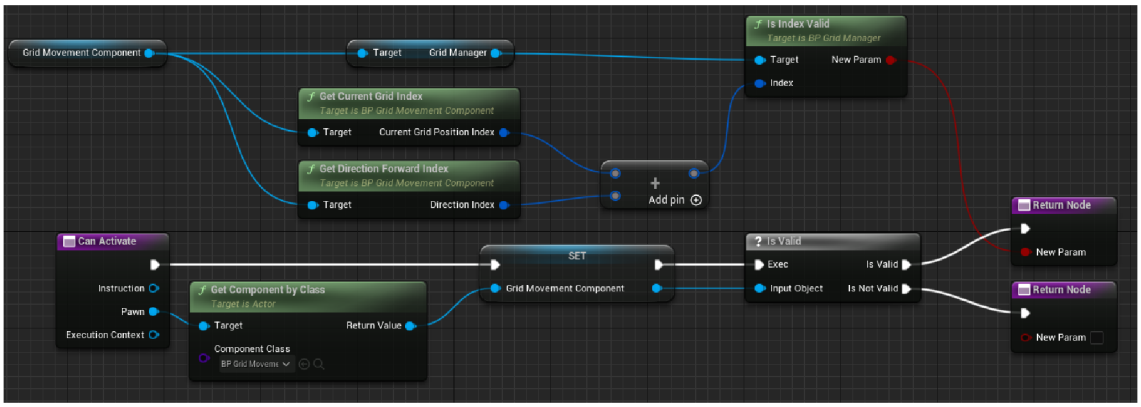
- **Podmínky v instrukcích**

Rozhodování v programování, typicky výraz *if*, vyžaduje podmínku, dle které se vykoná scénář *A* nebo scénář *B*. Právě proto vznikla třída *Conditional*, která má metodu *CanActivate*, díky které mohou instrukce rozpoznávat stav pravdy a nepravdy.

Conditional
- Name: Text +CanActivate(Instruction* Instruction, Pawn* CurrentPawn, InstructionExecutionContext* Context): bool

UML Class Diagram 5 Popis třídy podmínky v instrukcích (zdroj: vlastní)

Na obrázku níže je příklad implementace podmínky, která rozhoduje o tom, zda postava může udělat krok vpřed, aniž by opustila herní grid. O gridu více na straně č. 46.



Obrázek 15 Příklad definování instrukční podmínky – Krok vpřed (zdroj: vlastní)

Ve shrnutí se děje: Zjistí se, zda má hráčova postava *GridComponent*, pokud nemá, tak se vrací *false*, tedy nepravda. Jinak se zjistí momentální pozice hráče na gridu, přičte se k ní o pozice o jednu vpřed a následně se zjišťuje, zda je sečtená pozice validní. Tento výsledek se poté vrací.

Pro představu použití je zde stručně popsána implementace třídy *Instruction_Condition,* která dědí z *Instruction* třídy.

Instruction_Condition
+ ThenBuffer: InstructionBuffer*
+ ElseBuffer: InstructionBuffer*
+ Conditional: Conditional*

UML Class Diagram 6 Popis třídy Instruction_Condition (zdroj: vlastní)

Z diagramu lze chápat, že obsahuje dva InstructionBuffer objekty. Vykonání této instrukce závisí na výsledku *CanActivate* metody z *Conditional* objektu. Na základě toho bude spuštěna instrukce *ThenBuffer* nebo *ElseBuffer*.

- **Vytvoření instancí instrukcí**

Jelikož každá instrukce může mít jiné parametry než ostatní, tak každá instrukce v hierarchii je vlastním objektem. Pro vytvoření nové instance třídy *Instruction* se používají widgety, konkrétně widget vytvořený z atributu MenuDisplayClass, který při tažením myši vytvoří nový widget z DragInstructionDisplayClass a samotnou instanci třídy *Instruction*. Následně se o samotné přiřazení do hierarchie UI postará *DroppableArea* widget.

- **Inicializace herního prostředí**

Jak již bylo v předminulé kapitole zmíněno, vše týkající se inicializace herního prostředí obstarává *GameDescriptor* objekt v *GameInstance* objektu.
Postup inicializace je následovný:

1. V *GameMode* objektu se získá herní deskriptor z *GameInstance* objektu, v případě, že je nulový, tak se vytvoří staticky nastavený. Toto se děje v rámci vývoje aplikace, jelikož *GameDescriptor v GameInstance* je vždy nulový. Ve hře se toto nastaví po vybrání herní úrovně a po následném přesunu na mapu.
2. Na mapě se zjistí bod, kde má být postava vytvořena.
3. Získá se třída postavy z *GameDescriptoru*, vytvoří se a přesune na zjištěný bod.
4. Získá se *PlayerController*, přetypuje se a volá se metoda inicializace.

4.1. V této metodě se předají informace o dostupných instrukcích, kondicionálech a možnosti tvoření vlastních funkcí

4.2. Na základě těchto informací se inicializuje UI

V projektu lze najít následující typy *GameMode* tříd:

- BP_GameMode_MainMenu – Je použit v hlavní nabídce.
- BP_GameMode_GameDescriptor – Je použit primárně pro dědění.
- BP_GameMode_Tasks – Dědí z *BP_GameMode_GameDescriptor* a obsahuje možnost přidání vlastních herních úkolů.

A herních kontrolérů (*PlayerController*):

- BP_PlayerController_MainMenu – Použit v herní nabídce, sám o sobě nerozšiřuje nijak funkcionalitu. Pouze zobrazí kurzor.
- BP_PlayerController_Instructionable – Je použit v herních úrovních kde se pracuje s instrukcemi. Vytváří kořenový *InstructionBuffer* objekt, do kterého jsou následně přidávány další instrukce přes UI. Tvoří odpovídající uživatelské rozhraní podle *GameDescriptor* objektu. Vytvoří všechny *Conditional* objekty a uloží do pole *AvailableConditionals*.
- BP_PlayerController_Instructionable_IntructionTasks – Dědí ze třídy *BP_PlayerController_Instructionable* a rozšiřuje uživatelské rozraní o zobrazení daných herních úkolů.

- **Zobrazení instrukčních podmínek v UI**

V minulé kapitole bylo dáno, že herní kontrolér *BP_PlayerController_Instructionable* inicializuje *Conditional* objekty. Díky tomu lze ve widgetech přetypovat *PlayerController*, který se vrací z vestavěné metody *GetOwningController()* na již zmíněný, který obsahuje informace o *Conditional* objektech.

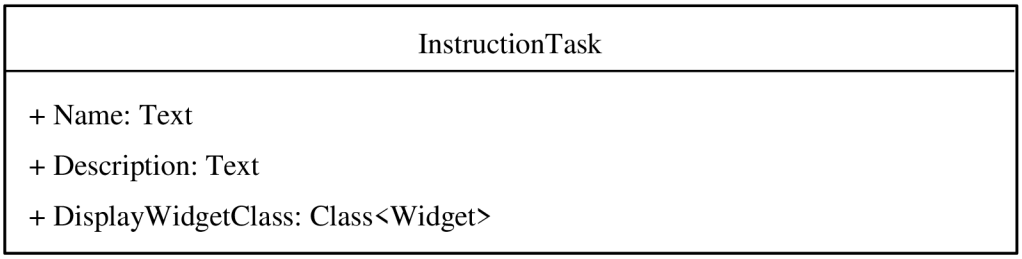
V kapitole o třídě *Instruction* bylo zmíněno, že každá instrukce má v sobě informaci o widgetech, které má zobrazit. Pro instrukce typu *Instruction_Condition*, lze definovat speciální widget, který bude dedikovaný pouze pro specifickou instrukci.

Ve widgetu bude vždy zřejmé, jak instrukci přetypovat, tak aby se mohly nastavovat libovolné parametry. Třeba v instrukci *Instruction_Condition* nastavovat *Condition* objekt a tím upravovat její podmínku.

- **Zpracování herních úkolů**

Pro zpracování herních úkolů byla vytvořena speciální třída *InstructionTask*.

UML class diagram by mohl vypadat následovně:



UML Class Diagram 7 Popis třídy úkolu (zdroj: vlastní)

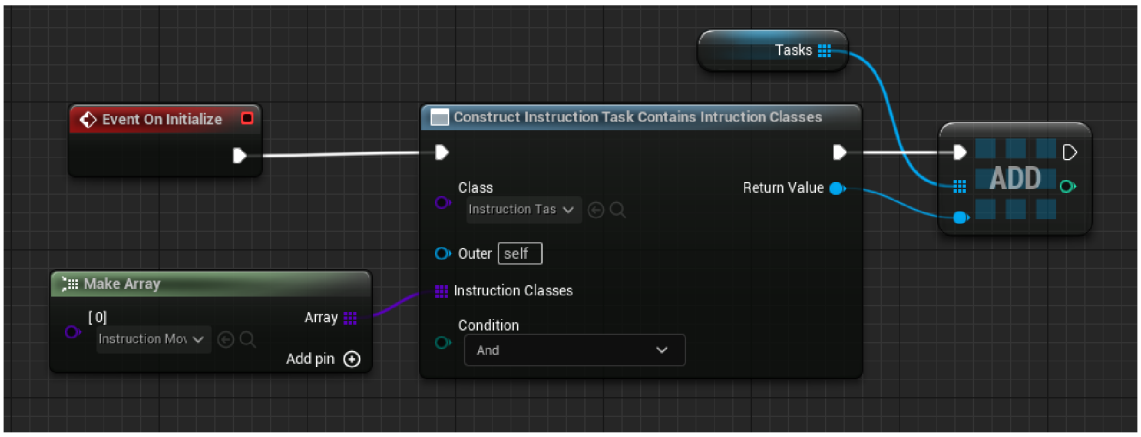
V této třídě je jen název úkolu, popis úkolu, popř. widget pro dodatečné zobrazení informací. Je zde i metoda *IsCompleted* pomocí které je možné zjistit, zda je úkol splněn.

Z této třídy následně dědí další úkoly jako:

- *InstructionTask_ContainsIntructionClasses* – Obsahuje pole specifických tříd instrukcí, které by měla hierarchie instrukcí obsahovat. Rekurzivně prochází instrukce a vyhodnocuje tuto situaci.
- *InstructionTask_GridIndex* – Hráčova postava musí stát na specifické pozici z gridu. Více v následující straně.

- **Definování herního úkolu**

Aby bylo možné zpracování v rámci herního úkolového módu, tak je nutné úkol přidat do pole *Tasks* v herním deskriptoru speciální třídy *InstructionGameDescriptor_Task*, která dědí ze třídy *GameDescriptor*. Následně budou úkoly přidány i do uživatelského rozhraní. Lze tak učinit pomocí přetížení *OnInitialize* eventu, jak je tomu na obrázku na další straně.



Obrázek 16 Blueprint definice herního úkolu (zdroj: vlastní)

Na obrázku lze vidět přidání herního úkolu *InstructionTask_ContainsIntructionClasses*, který udává, jaké instrukce musí být použity pro splnění úkolu. Nejprve se přetěžuje metoda *OnInitialize*, následně se vytvoří instance již zmíněného úkolu a poté se přidá do pole *Tasks*. Též lze na obrázku vidět několik možností, které zmíněný úkol *InstructionTask_ContainsIntructionClasses* má, např. uživatel musí použít více instrukcí najednou nebo třeba alespoň jednu z nich. Právě toto určuje podmínka *Condition*, která může nabývat stavu:

- Musí obsahovat všechny (*and*)
- Musí obsahovat alespoň jednu (*or*)

Následně je nutné definovat instrukce, které mají být použity. Pro definování jednotlivých instrukcí stačí nastavit vstupní parametr *InstructionClasses*, což je pole tříd instrukcí.

Následné vykreslení je zobrazeno na obrázku níže.

Obrázek 17 Zobrazení herních úkolů v UI (zdroj: vlastní)

Úkoly jsou vyhodnocovány až ve chvíli, kdy je kořenová instrukce u konce. V případě splnění se čtverečky zbarví do zelena, a tím je uživatel informován o jejich splnění.

- **Grid system**

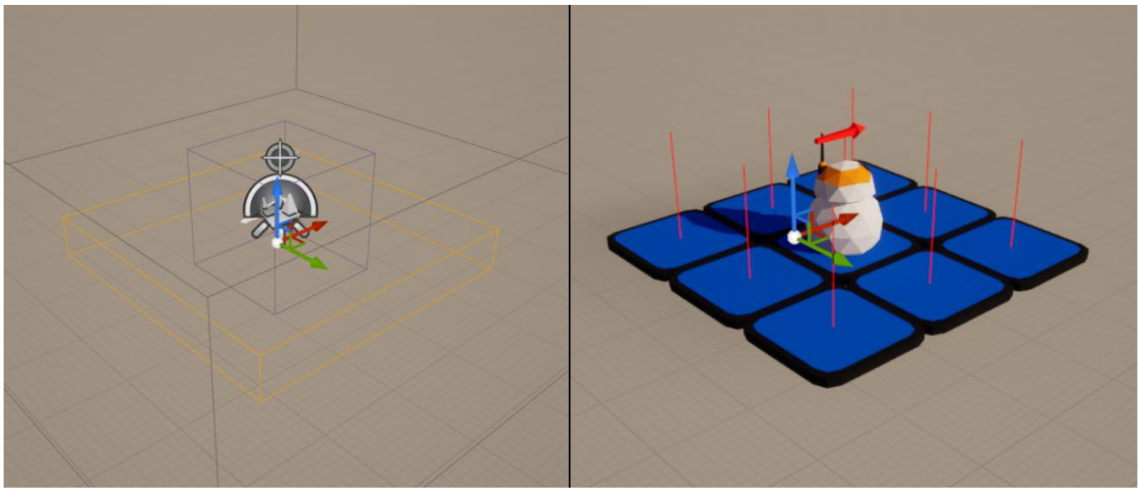
Celkový pohyb postavy hráče se pohybuje po tzv. gridu (mřížce, či 2D šachovnici). Tento styl ovládání se používá hlavně u tahových strategií.

Celý systém se dělí skládá ze tří tříd: *GridManager*, *GridBounds*, *GridRenderer*.

GridManager je objekt v herním světě, který je vždy unikátní a je pouze jeden.
Drží informace o jednotlivých pozicích (indexech) a jejich pozicích v herním světě.
Např. pozice (10;0) je v herním světě na lokaci (300;0;0).
Též obsahuje informace o tom, jak je veliká mřížka. Pomocí tohoto objektu lze zjistit, zda je pozice validní, nejbližší pozici k bodu v herním světě apod. Objekt pouze obsahuje informace, o již zmíněných datech a nic nevykresluje.

Pro definování jednotlivých pozic je možné do herního světa vložit objekt *GridBounds*, který je sám jakýmsi boxem. Tento box může následně uživatel různě škálovat a měnit jeho pozici, na základě toho budou vypočítány pozice uvnitř.
Každý tento objekt obsahuje i třídu/instanci *GridRenderer*, který se inicializuje pomocí metody *GetGridRenderer*, která vždy vrátí objekt na bázi *GridRenderer* a umožňuje vykreslovat různé druhy podstav (herních čtverců).

Samotnou kalkulaci pozic uvnitř tohoto boxu kalkuluje právě tento objekt.



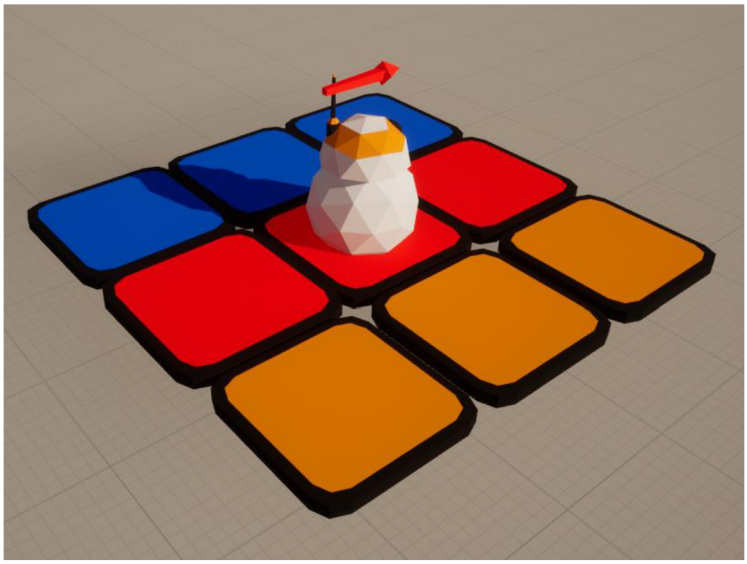
Obrázek 18 GridBounds objekt v herní scéně (zdroj: vlastní)

Na obrázku vlevo je žlutě vyznačený box, to je právě *GridBounds* objekt. Na pravé straně obrázku je spuštění hry, kdy je v herní scéně vytvořena mřížka uvnitř právě již zmíněného *GridBounds* objektu.

Sestavení samotného gridu v rámci objektu *GridManager* probíhá následovně:

1. V herním světě se najdou všechny *GridBounds* objekty.
2. Z každého objektu se vykalkulují validní pozice. Validní pozicí se myslí taková, která je na zemi. Toto je kontrolováno pomocí tzv. *ray castingem*, kdy je poslán paprsek z bodu A do budu B a kontroluje se, zda se protnul s nějakým objektem. Tato detekce je zjištěna na základě nastavené kolize objektů. Na předchozím obrázku je toto znázorněno na pravé straně pomocí červených linek, které jsou vidět pouze v editoru.
3. Vykreslí se pomocí *GridRenderer* objektu.
4. Přidá se do *GridIndices* proměnné.

Díky této variabilitě je možné tvořit různé nezávislé typy podstav.



Obrázek 19 Tři různé typy platforem ve hře (zdroj: vlastní)

V obrázku výše byly do herního světa přidány tři různé *GridBounds* objekty a každý z nich měl nastavené jiné vykreslování podstav. Ačkoliv se může zdát, že se změnila jenom barva, lze měnit i jeho relativní pozici, rotaci anebo třeba velikost, jak je tomu v případě oranžových podstav o cca 5 %.

- **Pohyb po mřížce**

Herní postava obsahuje komponentu *GridMovementComponent*, díky kterému lze postavou pohybovat po již zmíněném gridu. Tato komponenta úzce spolupracuje s komponentou *CharacterMovementComponent*, kterou je nutné mít na stejném objektu. *CharacterMovementComponent* je komponenta, která umožňuje třídám děděným z *Pawn* se pohybovat. (Unreal Engine) Více o této komponentě v oficiální dokumentaci Unreal Enginu.

Komponenta jenom a pouze přidává funkcionalitu pohybu, který je vázaný na pozice v *GridManager* objektu v herním světě. Díky této komponentě lze postavu *připnout* ke gridu a pohybovat se po něm. Obsahuje metodu *MoveToIndex(IntVector2D index)*, která zkontroluje lokaci v *GridManager* objektu a případně hne s postavou na určenou pozici. Též lze zjistit aktuální pozici, či pozici před herní postavou (myšleno *krok vpřed*).

Právě tato komponenta je používána v instrukci *MoveForward*. Více na straně: č. 32.

8.4. Struktura projektu a herních objektů/souborů

Vždy je doporučené si definovat takovou strukturu projektu, aby se v tom jak programátor, tak i ostatní dobře orientovali. Stromová struktura projektu vypadá následovně (v adresáři *Content*):

- **Blueprints**

V tomto adresáři jsou jen a pouze definované blueprinty. Je to tedy jediný adresář, kde se vyskytují. Je zde uvedena pouze jedna úroveň a popis, který obecně popisuje obsah.

- Components – V tomto adresáři jsou actor komponenty, jako již zmíněný GridMovementComponent.
- Environment – Zde se většinou vykytují actory, které jsou umístěné v herním světě (v tomto případě jsou zde definovány například světla).
- Functions – Blueprint funkce, které jsou sdíleny pro celý projekt (GridFunctions, MathFunctions, atd.).
- GameDescriptors – Zde je prostor pro jednotlivé třídy na bázi *GameDescriptor*.
- GameModes – Herní módy, které se v průběhu hry mění, např. hlavní nabídka, úkolový mód, či sandboxový mód.
- Grid – Třídy potřebné pro tvorbu gridu v herním světě.
- InstructionTasks – Veškeré actory pro tvorbu herních úkolů v úkolovém herním módu.
- Instructions – Třídy instrukcí, instrukčních výjimek a instrukčního call stacku. Též jsou zde implementovány jednotlivé instrukce.
- Level – Actory, které jsou používány v herním světě.
- Macros – UE makra, které se používají v celém projektu.
- PlayerControllers – Kontroléry pro různé již zmíněné herní módy.

- **Data**

V tomto adresáři jsou globální statická data jako struktura definice herní úrovně a jejich popis.

- **Levels**

Zde jsou herní mapy jako hlavní menu, nekonečná rovina (pro sandbox).

  - Testing – Mapy pro testování určitých herních funkcionalit.
  - GameLevels – Mapy pro módy s herními úrovněmi.

- **MaterialLibrary**

Materiály používané v rámci celého projektu, které jsou aplikované na herní postavu, herní svět, UI a další objekty.

- **Meshes**

Meshe (zpracovaný 3D objekt/model v rámci enginu) používané v projektu.

  - Characters – Herní postavy.
  - Environment – Prostředí.
  - Grid – Meshe používané pro grid, např. podstava herní pozice (čtverec v šachovnici).

- **Textures**

Textury (obrázky) používané v celém projektu. Jsou zde miniatury herních úrovní, UI prvky apod.

- **Widgets**

Jsou zde widgety pro tvorbu UI v celém projeku.

- Components – Univerzální widgety (např. *DroppableArea*).
- Fonts – Používané fonty v UI.
- Game – Používané widgety výhradně používané pro tuto hru.
  - Components – Univerzální widgety jako záložky orientované pouze na tento projekt
  - InGame – Widgety, které jsou používané mimo hlavní nabídku, tj. zobrazení úkolů, nástroje pro manipulaci s instrukcemi v UI (již zmíněná hierarchie).
  - MainMenu – Widgety pro funkčnost hlavní nabídky.
- Instruction – Různé widgety pro zobrazení instrukcí (jak položených bloků, tak i nabídku).

8.5. Pojmenovávání herních souborů

Pojmenovávání herních souborů je též velice důležité pro udržení konzistentnosti.

Projekt vychází z oficiální dokumentace Unreal Enginu. Většinou se jedná o styl pojmenovávání, přípony a předpony.

Zde jsou nějaké příklady, které se vyskytují v rámci projektu:

Předpona	Přípona	Význam
BP_		Blueprint
M_		Materiál
	_AnimBP	Animační blueprint
T_		Textura
F_		Struktura
BPI_		Blueprint rozhraní (interface)

Tabulka 1 Praktický projekt – pojmenovávání herních souborů (zdroj: vlastní)

8.6. Řízení vývoje projektu

Každý projekt v oblasti programování postupně zvětšuje svůj objem. V případě práce ve více lidech může být obtížné dobře spravovat kód, zvláště aby byl konzistentní.

Proto pro byl pro správu projektu zřízen veřejný GitHub repozitář. Lze jej najít na stránce:

HTTPS://GITHUB.COM/PETRVAVRINEK/BACHELOR-THESIS

Díky tomu lze procházet jednotlivé změny, či si projekt upravit podle sebe. Unreal Engine samotný má plugin, který umí efektivněji pracovat s git repozitáři, který byl využíván v průběhu implementace.

9. Shrnutí výsledků

V rámci této práce byl představen proces vývoje her rozdělený na čtyři různé části, které byly dále rozvedeny. Též byl rozveden pojem herní engine a představeny dnešní hojně využívané herní enginy se zaměřením na Unreal Engine. Také byl uveden rozdíl mezi enginem a editorem, proces získání enginu a vybraná funkcionalita. Další kapitola se věnovala tématu vizuální programování, kde byly rozebrány důvody proč se používá, výhody, nevýhody a přímé využití v praxi. Též zde byl věnován čas na propojení této metodiky programování s Unreal Enginem.
Dále byla rozebrána problematika využití her ve vzdělání a byly uvedeny jejich pozitivní i negativní stránky.

V praktické části práce byla demonstrována tvorba vzdělávací aplikace v Unreal Engine. V rámci této kapitoly byla rozebrána samotná analýza, návrh a následná implementace. Během části práce věnované implementaci byl popsán proces vytváření herních prvků, rozdělení práce na jednotlivé části a samotné provedení implementace.

Výsledkem praktické části práce je funkční vzdělávací aplikace vytvořená v Unreal Engine, která umožňuje uživatelům rozvíjet logické myšlení.

10. Závěry a doporučení

V práci bylo zjištěno, že použití Unreal Engine pro vývoj herních aplikací má velký potenciál a může být přínosem pro začátečníky v oblasti programování her díky vizuálnímu programování. Pokud se chcete zabývat použitím herních prvků ve vzdělávání, je důležité zvážit cíle aplikace a jak lze herní prvky integrovat do vzdělávacího procesu. Aplikace by měla být uživatelsky přívětivá, esteticky působivá a poskytovat užitečné informace pro uživatele. Celkově lze říct, že použití herních enginů, zejména Unreal Engine, může být výhodným nástrojem pro modernizaci vzdělávacích metod a přístupů. Pokud chcete mít hlubší znalosti vývoje her, zejména v Unreal Enginu, doporučuje se tvorba 3D modelů, nebo třeba vyzkoušení nativního C++ API pro rozšíření znalostí a funkcionalit enginu. Výhodou je určitě i návrh UX pro efektivnější zpracování uživatelského rozhraní.

Seznam použité literatury

Bellincampi, Lorenzo. Game Instance in Unreal Engine 4. *PixelSapiens.* [Online] [Citace: 25. Březen 2023.] https://pixelsapiens.com/game-instance-in-unreal-engine-4/.

Bramble, Ross. 2023. What Are The Main Stages Of Game Development? *GameMaker.* [Online] 4. Leden 2023. https://gamemaker.io/en/blog/stages-of-game-development.

BroadbandSearch. Mobile Vs. Desktop Internet Usage (Latest 2023 Data). *BroadbandSearch.* [Online] https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics.

Denham, Thomas. What are 3D & Game Shaders? *Concept Art Empire.* [Online] https://conceptartempire.com/shaders/.

Fuentes, Lauren. 2021. Blender: Low-Poly Design – Simply Explained. *All3DP.* [Online] 9. Listopad 2021. [Citace: 25. Březen 2023.] https://all3dp.com/2/blender-low-poly-tutorial/.

Fuka, Jiří. Zvyšují hry lidskou agresi? *PLAYzone.cz.* [Online] https://www.playzone.cz/clanky/184646-zvysuji-hry-lidskou-agresi.

Huivan, Nataly. 2022. What Is Visual Programming? *GudHub Platform.* [Online] 16. Leden 2022. https://gudhub.com/blog/low-code-no-code/visual-programming/.

Indeed Editorial Team. 2022. What Is Game Publishing? (Definition, Duties and Companies). *Indeed.com.* [Online] 1. Říjen 2022. https://www.indeed.com/career-advice/career-development/what-is-game-publishing.

Janovský, Dušan. Gamesy. *Yuhů = Dušan Janovský.* [Online] https://dusan.pc-slany.cz/archiv/pocthrypsych.htm.

Kadlec, Petr. Petr Kadlec -- homepage. *Petr Kadlec -- homepage.* [Online] http://mormegil.wz.cz/prog/index.html.

Kašpar, Radek. 2020. Vzdělávací aspekt počítačových her. *Medium.* [Online] 2020. Květen 2020. https://medium.com/edtech-kisk/vzd%C4%9Bl%C3%A1vac%C3%AD-aspekt-po%C4%8D%C3%ADta%C4%8Dov%C3%BDch-her-402612b80c03.

Krejčí, Martin. Počítačové hry a studium angličtiny. *Anglická slovíčka.* [Online] https://www.ajslovicka.cz/clanky/pocitacove-hry-a-studium-anglictiny.html.

Leška, Šimon. 2020. Co je to UX a UI design? *BlueGhost.* [Online] 13. Listopad 2020. https://www.blueghost.cz/clanek/co-je-to-ux-a-ui-design/.

MDN. Call stack - MDN Web Docs Glossary: Definitions of Web-related terms. *MDN.* [Online] [Citace: 22. Březen 2023.] https://developer.mozilla.org/en-US/docs/Glossary/Call_stack.

MEMOS. Prototyp. *MEMOS Software | Programátorsko-český slovník.* [Online] https://www.memos.cz/slovnik/prototyp/.

Merheb, Ahmad. 2023. Best Game Engine of 2023 (Ranked and Reviewed). *Ahmad Merheb - 3D Animation, Gaming and Productivity.* [Online] 9. Duben 2023. https://ahmadmerheb.com/best-game-engine/.

Ossian. 2022. What Is Visual Programming & Why Is It Important? A General Guide. *DataMyte.* [Online] 24. Listopad 2022. https://www.datamyte.com/visual-programming/.

Pařízek, Václav. Co je to storyboard a jak vypadá? *zabavnamedia.tv.* [Online] https://zabavnamedia.tv/co-je-to-storyboard-a-jak-vypada/.

Pešek, Ondřej. 2022. Co je to herní engine? *Zboží.cz.* [Online] Červenec 2022. https://www.zbozi.cz/magazin/c/co-je-to-herni-engine/.

Scratch. Scratch foundadion. *Scratch.* [Online] https://scratch.mit.edu/.

Unity. Unity. *Unity.* [Online] https://unity.com/.

Unreal Engine. Frequently Asked Questions. *Unreal Engine.* [Online] [Citace: 22. Březen 2023.] https://www.unrealengine.com/en-US/faq.

—. Unreal Engine 5.1 Documentation. *Unreal Engine 5.1 Documentation.* [Online] https://docs.unrealengine.com/5.1/en-US/.

Verma, Saumya. 2022. Introduction to Visual Programming in Architecture (2022). *Oneistox.* [Online] 8. Listopad 2022. https://www.oneistox.com/blog/visual-programming-in-architecture.

Seznam obrázků

Seznam tabulek

Seznam UML Class diagramů

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Petr Vavřínek**
Studium:	I1900272
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Edukativní aplikace s využitím Unreal Engine**
Název bakalářské práce AJ:	Educational application using Unreal Engine

Cíl, metody, literatura, předpoklady:

Cíl práce
Vytvoření edukativní aplikace s použitím Unreal Enginu

Osnova

1. Úvod
2. Cíl práce
3. Metodika vypracování
4. Vývoj her (Herní engine, Unreal Engine, Alternativní herní enginy)
5. Vizuální programování
6. Využití her ve vzdělávání
7. Analýza a návrh aplikace
8. Implementace aplikace
9. Závěr a doporučení

An Introduction to Unreal Engine 4 | Andrew Sanders | Taylor & Francis. Home | Taylor & Francis eBooks, Reference Works and Collections [online]. Copyright © 2022 Informa UK Limited [cit. 12.10.2022]. Dostupné z: https://www.taylorfrancis.com/books/mono/10.1201/9781315382555/introduction-unreal-engine-4-andrew-sanders

Výzkumy o použití videoher ve vzdělávání - Game Based Learning. [online]. Dostupné z: https://sites.google.com/a/m77.cz/game-based-learnin/vyzkumy-o-pouziti-videoher-ve-vzdelavani

GAME ENGINES IN SCIENTIFIC RESEARCH | Michael Lewis and Jeffrey Jacobson [online]. Copyright © [cit. 12.10.2022]. Dostupné z: https://www.cse.unr.edu/ sushil/class/gas/papers/GameAIp27-lewis.pdf

Současné počítačové hry a jejich možnost implementace do výuky coby novodobý prostředek rozvoje žáků | Terezie Otrubová [online]. Copyright © [cit. 12.10.2022]. Dostupné z: https://is.muni.cz/th/xr2sh/Diploma_Thesis_digital_Archive.pdf?lang=cs

VOJTĚCH, Petr. Možnosti využití počítačových her ve výuce společenskovědních předmětů [online]. Hradec Králové, 2017 [cit. 2022-10-12]. Dostupné z: https://theses.cz/id/0t2wqn/. Bakalářská práce. Univerzita Hradec Králové, Pedagogická fakulta. Vedoucí práce Mgr. Jana Andršová.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D.
Datum zadání závěrečné práce:	21.1.2020