|  |  |  |
| --- | --- | --- |
|  | | |
| **Závěrečná studijní práce**  **Dokumentace** | | |
| **Hranice prostoru a perspektivy ve videohrách** | | |
| Lukáš Olbrecht | | |
|  | | |
|  | |  |
| **Obor:** | 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  se zaměřením na počítačové sítě a programování | |
| **Třída:**  **Školní rok:** | IT4  2023/2024 | |

**Poděkování**

*Rád bych vyjádřil své poděkování panu Markovi Lučnému za cennou zpětnou vazbu a podporu, kterou mi poskytoval během tvorby této závěrečné práce.*

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité   
informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové   
a umělecké škole v Opavě, Praskova 399/8.

V Opavě 31. 12. 2023

*podpis autora práce*

**ABSTRAKT**

Tento projekt se zabývá implementací optických klamů inspirovaných hrou Superliminal. Projekt je realizován ve dvou různých herních enginech – Unity a Unreal Engine.  
V Unity je vytvořena hlavní mechanika hry Superliminal, která je známá jako Forced Perspective. Tato technika využívá perspektivu hráče k vytvoření iluze změny velikosti objektů. Například při pohledu z určitého úhlu může objekt vypadat menší nebo větší, než ve skutečnosti je, a tento efekt je následně zachován i při interakci s objektem.

Unreal Engine byl naopak použit pro implementaci systému neviditelných portálů. Tyto portály umožňují plynulý pohyb hráče mezi prostory, aniž by zaznamenal, že byl ve skutečnosti přenesen na zcela jiné, vzájemně nenavazující místo. Portály lze dále využít pro změnu velikosti hráče či vytvoření tzv. nekonečné chodby, kde hráč prochází opakující se sekvencí prostorů.

**ABSTRACT**

This project deals with the implementation of optical illusions inspired by the game Superliminal. The project is implemented in two different game engines - Unity and Unreal Engine.

The main game mechanic of Superliminal is created in Unity, which is known as Forced Perspective. This technique uses the player's perspective to create the illusion of resizing objects. For example, when viewed from a certain angle, an object can appear smaller or larger than it actually is, and this effect is then maintained when interacting with the object.

Unreal Engine, on the other hand, was used to implement the system of invisible portals. These portals allow the player to move seamlessly between spaces without noticing that they have actually been transported to a completely different, unrelated location. Portals can also be used to change the size of the player, which enables effects such as enlarging, shrinking or creating so-called endless corridors where the player goes through a repeating sequence of spaces.

OBSAH

[Úvod 5](#_Toc155451207)

[1 Teoretická a metodická východiska 6](#_Toc155451208)

[1.1 Text první kapitoly 6](#_Toc155451209)

[1.1.1 Třetí úroveň 6](#_Toc155451210)

[2 Využité technologie 7](#_Toc155451211)

[3 Způsoby řešení a použité postupy 8](#_Toc155451212)

[4 Výsledky řešení, výstupy, uživatelský manuál 9](#_Toc155451213)

[Závěr 10](#_Toc155451214)

[Seznam použitýCH INFORMAČNÍCH ZDROJů 11](#_Toc155451215)

[Seznam příloh 12](#_Toc155451216)

Úvod

* Tato práce se zabývá implementací inovativních herních mechanik inspirovaných optickými klamy a portálovými systémy, jak je známe například ze hry Superliminal. Cílem projektu je vytvořit funkční demonstrační aplikaci ve dvou předních herních enginech: Unity a Unreal Engine. Projekt se zaměřuje na dvě specifické mechaniky: Forced Perspective v Unity
* a portálové systémy v Unreal Engine.
* Herní mechaniky založené na perspektivě a portálech představují zajímavé vývojářské výzvy, které vyžadují hluboké pochopení 3D prostoru, práce s kamerou a iluzemi. Tyto prvky nejen obohacují herní zážitek, ale také posouvají hranice technických možností herních enginů. Inspirací pro práci byla hra Superliminal, která ukazuje potenciál optických klamů a změny měřítka v herním designu.

## Cíle práce

* Implementovat mechaniku Forced Perspective v Unity, která mění velikost objektů na základě hráčovy perspektivy.
* Vytvořit portálové systémy v Unreal Engine, které umožňují plynulý přechod mezi prostory, včetně efektů zvětšování, zmenšování a nekonečných chodeb.

1. Kapitola se věnuje teoretickým a metodickým východiskům, která objasňují základní principy použitých herních mechanik a postupů.
2. Kapitola popisuje technologie využité při implementaci, konkrétně enginy Unity a Unreal.
3. Kapitola se zaměřuje na způsoby řešení a detailně popisuje postupy použité při vývoji mechanik.
4. Kapitola prezentuje výsledky řešení, zahrnuje dosažené výstupy a obsahuje uživatelský manuál k vytvořeným mechanikám.

Závěr shrnuje hlavní poznatky práce a hodnotí dosažené cíle.

# Teoretická a metodická východiska

## Pohyb hráče v Unity Engine

Pohyb hráče je základním prvkem většiny her a jeho správná implementace ovlivňuje celkový zážitek z hry. V tomto projektu je pohyb hráče implementován pomocí dvou skriptů: PlayerMovement.cs a MouseLook.cs, které zajišťují ovládání pohybu postavy a pohledu hráče v prostředí Unity.

### **PlayerMovement.cs**

Skript PlayerMovement.cs je odpovědný za horizontální pohyb hráče, skok a gravitaci. Využívá komponentu CharacterController, která usnadňuje práci s pohybem v 3D prostoru bez nutnosti ruční implementace kolizí.

Horizontální pohyb: Pohyb hráče je řízen pomocí vstupů z klávesnice (Horizontal a Vertical) a je převeden na vektor pohybu ve světových osách X a Z. Pohyb je normalizován, aby byla rychlost konstantní i při pohybu úhlopříčně.

Skok je realizován na základě kontroly, zda se hráč nachází na zemi, což je ověřováno metodou Physics.CheckSphere. Pokud hráč stiskne tlačítko pro skok a je na zemi, vypočítá se vertikální rychlost na základě výšky skoku a gravitační síly.

Skript aplikuje konstantní gravitaci směrem dolů. Pokud hráč opustí povrch (např. po skoku), gravitace jej začne postupně zrychlovat směrem k zemi.

#### Ukázka klíčových funkcí skriptu

* Pohyb hráče na základě vstupu

float x = Input.GetAxisRaw("Horizontal");  
float z = Input.GetAxisRaw("Vertical");  
Vector3 move = transform.right \* x + transform.forward \* z;  
controller.Move(move.normalized \* speed \* Time.deltaTime);

* Implementace skoku

if (Input.GetButtonDown("Jump") && isGrounded)  
{  
 velocity.y = Mathf.Sqrt(jumpHeight \* -2f \* gravity);  
}

* Aplikace gravitace

velocity.y += gravity \* Time.deltaTime;  
controller.Move(velocity \* Time.deltaTime);

### MouseLook.cs

Skript MouseLook.cs zajišťuje ovládání pohledu hráče pomocí myši. Tento skript je klíčový pro vytvoření tzv. first-person perspektivy, která je typická pro hry z pohledu první osoby.

Ovládání pohledu: Pohyb myší ve vodorovném směru (osa X) otáčí celým tělem hráče, zatímco pohyb myší ve svislém směru (osa Y) ovládá otáčení kamery nahoru a dolů.

Clamping rotace: Rotace kamery ve vertikální ose je omezena metodou Mathf.Clamp, aby hráč nemohl otočit pohled příliš daleko za limity (např. za hlavu postavy).

Uzamknutí kurzoru: Pro plynulý pohyb pohledu je kurzor myši uzamčen do středu obrazovky pomocí Cursor.lockState.

#### Ukázka klíčových funkcí skriptu

* Ovládání pohledu myší

float mouseX = Input.GetAxis("Mouse X") \* mouseSensitivity \* Time.deltaTime;  
float mouseY = Input.GetAxis("Mouse Y") \* mouseSensitivity \* Time.deltaTime;

* Clamping a vertikální rotace

xRotation -= mouseY;  
xRotation = Mathf.Clamp(xRotation, -90f, 90f);  
transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);

* Otáčení hráčova těla

playerBody.Rotate(Vector3.up \* mouseX);

### **Spolupráce skriptů**

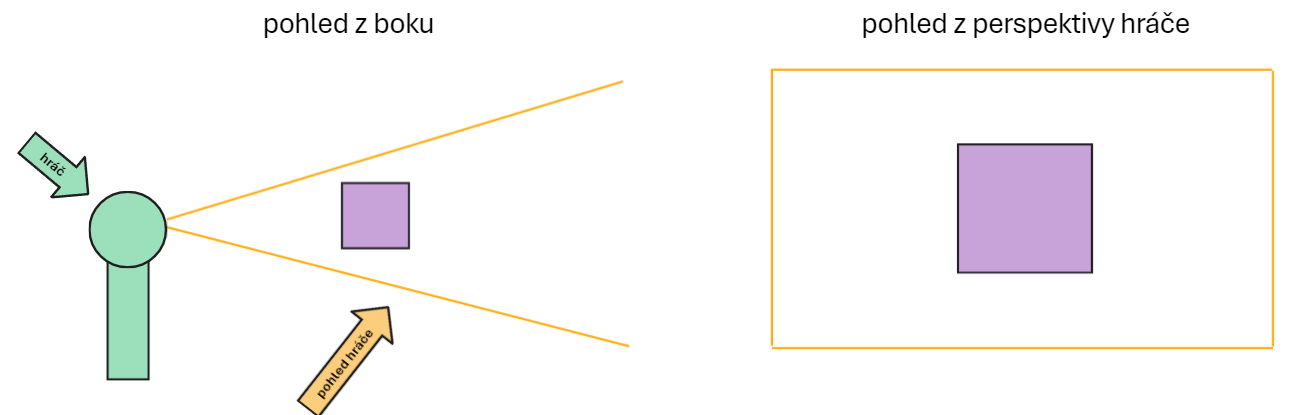
Skripty PlayerMovement.cs a MouseLook.cs spolupracují na vytvoření komplexního systému pohybu a ovládání pohledu hráče:

* PlayerMovement.cs zajišťuje pohyb a skákání hráče.
* MouseLook.cs ovládá pohled hráče a rotaci jeho těla podle pohybu myši.
* Díky kombinaci těchto skriptů je dosaženo plynulého ovládání postavy ve first-person perspektivě. Hráč může pohybovat postavou ve všech směrech, skákat a otáčet se po vertikální i horizontální ose.

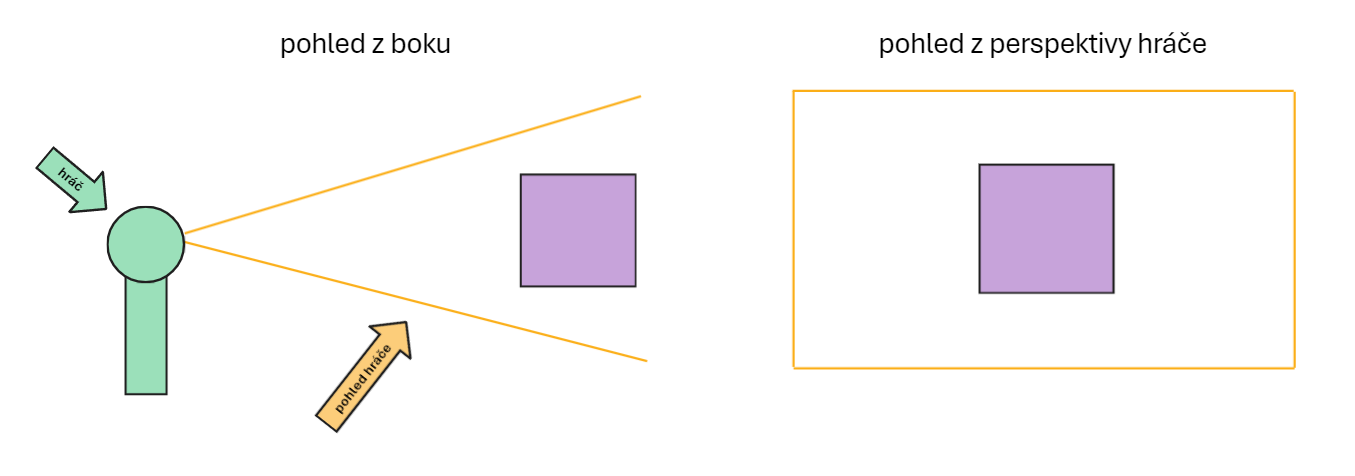
# **Forced perspective**

Jednou z klíčových mechanik hry *Superliminal* je tzv. **forced perspective** – efekt, který umožňuje měnit velikost objektů na základě perspektivy hráče. Tento princip spočívá v tom, že objekty zdánlivě zachovávají svou velikost i když jsou ve skutečnosti větší.

1. **Startovní pozice objektu**
   1. Představme si hráče, který drží objekt ve vzdálenosti 1 metru.
   2. Z této vzdálenosti objekt vypadá určitým způsobem – například vyplňuje určitou část zorného pole hráče.



1. **Změna vzdálenosti objektu**
   1. Nyní hráč posune objekt dále od sebe, například do vzdálenosti 2 metrů.
   2. Z pohledu hráče objekt stále vyplňuje stejnou část zorného pole, ale jeho velikost ve světě je dvojnásobná



**Jak to funguje?**

Z pohledu kamery (hráče) je klíčem k tomuto efektu zachování **úhlové velikosti objektu** ve zorném poli. Pokud je objekt dál, jeho skutečná velikost se zvětší, aby se vizuálně jevil stejně velký. Tento princip lze simulovat pomocí přepočtu vzdálenosti objektu od kamery a přizpůsobení jeho měřítka.

***Klíčové principy skriptu***

* Skript umožňuje hráči vzít objekt pohledem (raycastem) a přiřadit jej jako držený. Objekt se přichytí k hráči, deaktivuje svou fyziku (Rigidbody) a jeho velikost i pozice se přizpůsobí hráčově perspektivě.
* Při držení objektu se jeho pozice dynamicky upravuje tak, aby byl vždy před překážkami, což zabraňuje tomu, aby objekt kolidoval se zdmi nebo jinými překážkami v prostředí.
* Objekt mění svou velikost podle vzdálenosti od hráče, čímž simuluje efekt perspektivy. Pokud je objekt dále od hráče, zvětšuje se; pokud je blíže, zmenšuje se. Velikost se vypočítává na základě poměru původní velikosti objektu a vzdálenosti od hráče.
* Skript vytváří mřížku bodů (grid) na povrchu objektu, které jsou používány pro detekci kolizí s okolním prostředím. Tato mřížka umožňuje přesné přizpůsobení pozice objektu při držení hráčem, aby byl vždy volný prostor před ním.
* Skript používá viewport kamery k výpočtu nových pozic a orientace objektu, což zajišťuje, že objekt zůstane správně zarovnaný z pohledu hráče, ať už je vzdálený nebo blízký.

#### Ukázka klíčových funkcí skriptu

GetBoundingBoxPoints()

private Vector3[] GetBoundingBoxPoints()  
{  
 Vector3 size = heldObject.GetComponent<Renderer>().localBounds.size;  
 Vector3 x = new Vector3(size.x, 0, 0);  
 Vector3 y = new Vector3(0, size.y, 0);  
 Vector3 z = new Vector3(0, 0, size.z);  
 Vector3 min = heldObject.GetComponent<Renderer>().localBounds.min;  
  
 // Body bounding boxu  
 Vector3[] bbPoints =  
 {  
 min,  
 min + x,  
 min + y,  
 min + x + y,  
 min + z,  
 min + z + x,  
 min + z + y,  
 min + z + x + y  
 };  
  
 return bbPoints;  
}

Skript vytvoří obdélník okolo viditelné části objektu

Funkce SetupGrid()

private Vector3[,] SetupGrid()  
{  
 // Vypočítáme délky hran ve vodorovném a svislém směru  
 float rectHrLength = right.x - left.x;  
 float rectVertLength = top.y - bottom.y;  
  
 // Krok mřížky (horizontální a vertikální)  
 Vector3 hrStep = new Vector2(rectHrLength / (NUMBER\_OF\_GRID\_COLUMNS - 1), 0);  
 Vector3 vertStep = new Vector2(0, rectVertLength / (NUMBER\_OF\_GRID\_ROWS - 1));  
  
 // Inicializace mřížky  
 Vector3[,] grid = new Vector3[NUMBER\_OF\_GRID\_ROWS, NUMBER\_OF\_GRID\_COLUMNS];  
 grid[0, 0] = new Vector3(left.x, bottom.y, left.z);  
  
 // Generování bodů mřížky  
 for (int i = 0; i < grid.GetLength(0); i++) // Řádky  
 {  
 for (int j = 0; j < grid.GetLength(1); j++) // Sloupce  
 {  
 if (i == 0 && j == 0) continue;  
  
 if (j == 0) // První bod v řádku  
 {  
 grid[i, j] = grid[i - 1, 0] + vertStep;  
 }  
 else // Následující body v řádku  
 {  
 grid[i, j] = grid[i, j - 1] + hrStep;  
 }  
 }  
 }  
  
 return grid;  
}

Vytvoří síť bodů skrze které se později posílají raycasty.

Funkce MoveInFrontOfObstacles()

void MoveInFrontOfObstacles()  
{  
 float closestZ = 1000;  
  
 // Projdeme body mřížky a detekujeme překážky  
 foreach (Vector3 gridPoint in shapedGrid)  
 {  
 RaycastHit hit = CastTowardsGridPoint(gridPoint, wallLayer + pickUpLayer);  
 if (hit.collider == null) continue;  
  
 // Získáme bod nejbližší překážky  
 Vector3 wallPoint = transform.InverseTransformPoint(hit.point);  
 if (wallPoint.z < closestZ)  
 {  
 closestZ = wallPoint.z;  
 }  
 }  
  
 // Posun objektu o určitý offset od překážky  
 float boundsMagnitude = heldObject.GetComponent<Renderer>().bounds.extents.magnitude \* heldObjectTF.localScale.x;  
 Vector3 newLocalPos = heldObjectTF.localPosition;  
 newLocalPos.z = closestZ - boundsMagnitude;  
 heldObjectTF.localPosition = newLocalPos;  
}

Objekt posuneme před nejbližší překážku aby byl pořád v blízkosti hráče

# Pohyb hráče v Unreal Engine

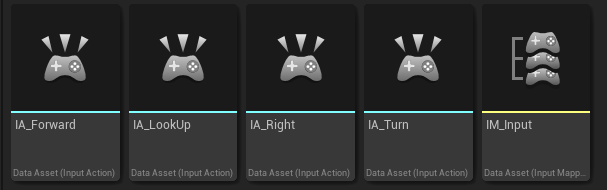
Pohyb hráče v Unreal Engine se výrazně liší od Unity především v přístupu k jeho implementaci. Zatímco v Unity je nutné pohyb hráče programovat skrze vlastní kód, v Unreal Engine jsou základní mechanismy, jako například chůze, přednastavené. Stačí je jednoduše nastavit a propojit.

Pro zajištění ovládání hráče je nutné vytvořit Input Mapping Context (IMC) a Input Action (IA).

* **Input Action (IA)** slouží jako prostředník mezi vstupy (například pohyb myši nebo stisk klávesy) a tím, co se má stát ve hře.
* **Input Mapping Context (IMC)** následně umožňuje přiřadit jednotlivým vstupům konkrétní akce. V IMC tedy rozhodujeme, co má každý vstup vykonat – například přiřazení klávesy „W“ k pohybu vpřed nebo pohybu myší k otáčení kamery.

Tento systém výrazně zjednodušuje práci s ovládáním a poskytuje větší flexibilitu při nastavování interakcí ve hře.

K jednoduchému pohybu nám tedy stačí IA jen 4x:



Uvnitř IMC nastavení pro chůzi dopředu a dozadu vypadá takto:

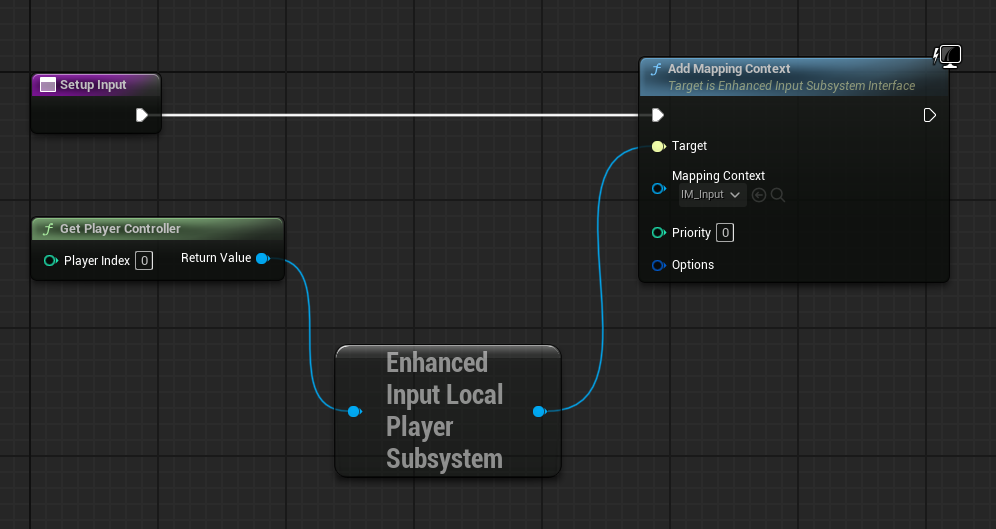
Jak je vidět na přiloženém obrázku, pohyb dopředu i dozadu využívá stejnou osu. Díky tomu stačí vytvořit pouze jedno IA, které obsluhuje oba směry pohybu. Pro zajištění pohybu dozadu je k této akci přiřazen modifikátor, který neguje hodnotu osy. Tento přístup zjednodušuje implementaci ovládání a minimalizuje množství potřebných akcí, protože umožňuje efektivně využít jednu osu pro více směrů.

Stejný princip je použit i pro pohyb do stran, tedy doprava a doleva. Pohyb oběma směry využívá stejnou osu, přičemž pohyb doleva je realizován negací hodnoty této osy pomocí modifikátoru. Tento jednotný přístup umožňuje snadnou správu vstupů a snižuje množství potřebných IA, což vede k jednodušší a přehlednější konfiguraci ovládání.

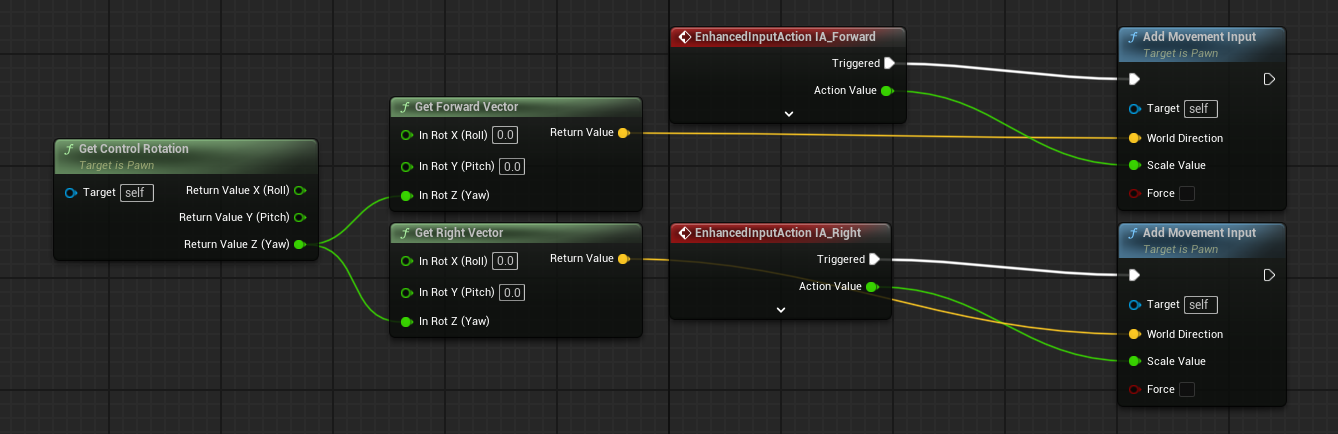
Toto ale není vše co je potřebné udělat proto abychom mohli pohy.ovat hráčem. V objektu hráče musíme vytvořit funkci která nám dovolí užívat náš IMC vytvoření této funkce je jednoduché.

To však není vše, co je potřeba udělat, aby bylo možné pohybovat hráčem. V blueprintu hráče je nutné vytvořit funkci, která umožní používat náš IMC. Tento krok je zásadní, protože právě tímto způsobem přiřazujeme náš IMC konkrétnímu hráči, aby mohl reagovat na vstupy.

Na následující straně je grafické znázornění této funkce, kde je jasně vidět propojení uzlů a logika přiřazení IMC hráči.



Nyní, když jsme mapping context úspěšně přiřadili hráči, můžeme definovat konkrétní akce, které se mají provést při aktivaci tohoto mapping contextu. To nám umožňuje propojit vstupy hráče, jako je pohyb, otáčení kamery nebo interakce, s odpovídajícími funkcemi nebo logikou ve hře.



# Neviditelné portály

Pro vytvoření portálu v Unreal Engine je potřeba sestavit objekt, který bude obsahovat několik klíčových komponent. Základem je jednostranná plocha (Plane), která slouží jako "plátno" pro promítání obrazu. Aby portál vypadal vizuálně přirozeněji, doporučuji kolem této plochy vytvořit rám. Ten slouží nejen jako estetický prvek, ale také může pomoci s materiálovým odlišením portálů, pokud jich má hra více.

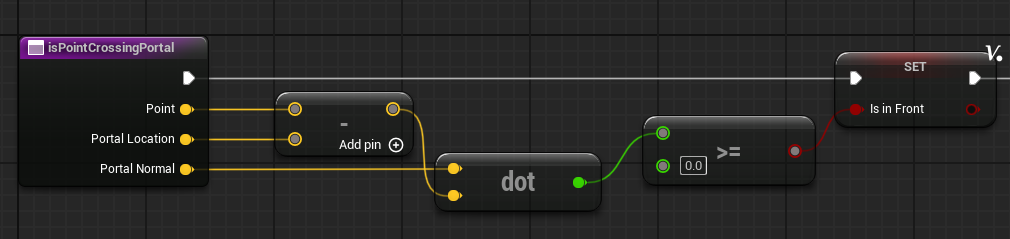
Důležitou součástí portálu je kamera, konkrétně Screen Capture Component 2D, která zachytává obraz scény z pohledu portálu. Tento obraz je následně promítán na druhý portál, což vytváří dojem, že hráč skutečně vidí skrze portál do jiného prostoru. Například portál\_1 využívá svou kameru k zachycení obrazu scény za portálem\_2. Obraz zachycený kamerou portálu\_1 se promítá na plochu portálu\_2, což je realizováno pomocí speciálního materiálu, který využívá texturu generovanou komponentou Render Target. Tímto způsobem dochází k dynamickému propojení obou portálů.

Aby bylo možné portálem plynule procházet, doporučuji upravit několik grafických nastavení Unreal Engine. Zaprvé je vhodné vypnout Lumen, který zajišťuje globální osvětlení, protože může způsobovat nežádoucí odrazy a nesprávné osvětlení uvnitř portálu. Dále je potřeba deaktivovat Global Illumination, aby se předešlo zbytečnému zatížení výkonu a nekonzistentním světelným efektům. Nakonec doporučuji vypnout Anti-Aliasing, čímž se eliminují možné vizuální artefakty na hranách portálu.

Výsledkem všech těchto kroků je portál, který vypadá jako skutečný průchod do jiného prostoru. Hráč při pohledu na portál vidí věrnou simulaci toho, co by se nacházelo na druhé straně, a při průchodu portálem je iluze zcela plynulá a bez vizuálních rušivých elementů. Tento postup zajišťuje, že portály nejen vypadají realisticky, ale také umožňují hráči bezproblémový přechod mezi různými částmi herního světa.



Pro teleportaci mezi portály použijeme skalární součin. Díky němu jsme schopni zjistit, na které straně portálu se hráč nachází, čímž se předejde nežádané teleportaci, pokud hráč projde stranou portálu, ze které není plátno viditelné.



Nakonec použijeme materiál pro plátno portálu, který vizuálně "prohne" plátno. Tento efekt je důležitý, protože při přímém pohledu na plátno během průchodu portálem by hráč mohl vidět druhou stranu, kterou právě prochází. Prohnutí plátna zakryje tuto nežádoucí scénu a zajistí, že hráč bude mít dojem plynulého přechodu do jiného prostoru. Tímto způsobem se udrží iluze, že portál je přímým průchodem mezi místy.

# Využité technologie

Text druhé kapitoly

* nástroje, přístroje, programy a jiné materiální prostředky, včetně zdůvodnění jejich výběru, použité při řešení zadaného úkolu
* informace o použitých technologiích by neměly mít charakter reklamy na daný výrobek, ale měly by přinášet objektivní technický popis použitých prostředků včetně zdůraznění a vysvětlení klíčových parametrů

# Způsoby řešení a použité postupy

Text třetí kapitoly

* popis řešení úkolu včetně, použité postupy a jejich vysvětlení, způsoby testování funkčnosti, parametry výrobku (programu, hotového řešení), schémata, obrázky z tvorby a finálního provedení, výpočty, použité příkazy…

# Výsledky řešení, výstupy, uživatelský manuál

Text čtvrté kapitoly

* výčet splněných a nesplněných cílů, obrázky (schémata, vzorce apod.) z finálního provedení, prokázání funkčnosti, výsledné parametry výrobku apod.
* podle zaměření a charakteru práce je třeba volit vhodný nadpis pro tuto kapitolu, je samozřejmě možné i rozdělení na více kapitol (např. Uživatelské rozhraní internetové aplikace; Administrace internetové aplikace…)

# Závěr

Text závěru

* povinná část,
* shrnuje výsledky, hodnotí splnění cíle práce, uvádí možnost uplatnění řešení v praxi a nastínění případných dalších budoucích vylepšení
* kapitola se nečísluje (stejné jako úvod)

Seznam použitýCH INFORMAČNÍCH ZDROJů

[1] BOHMAN, Ludvík. Zákon o pojistné smlouvě. Praha: Linde Praha a. s., 2004. 381 s. ISBN80-7201-504-4

[2] DUCHÁČKOVÁ, Eva. Principy pojištění a pojišťovnictví. 3. aktualizované vydání. Praha: Ekopress 2009. 224 s. ISBN 978-80-86929-51-4

[3] KUBALA, Petr. Planetární dvojcata - Věda a technika (Český rozhlas) [online].   
Č. 2000-2008, poslední revize 19. 3. 2008 [cit. 2008-03-20].  
<http://www.rozhlas.cz/veda/vesmir/\_zprava/435849>.

[4] KULDOVÁ, O., FLEISCHMANNOVÁ, E. Metodická příručka k technice administrativy a obchodní korespondence. 1.vyd. Praha: Fortuna 1998. 111 s.   
ISBN 80-7168-574-7. Kapitola 6, Metody nácviku psaní hmatovou metodou,   
s. 28-29.

[5] VLACH, J. JE Temelín a zásobování teplem. Energetika, 2001, roč. 51, č. 3, s. 84 -85. ISSN 0375-8842.

* musí zahrnovat všechny prameny, knihy, internetové odkazy a další studijní podklady, z nichž jsme čerpali;
* kapitola se nečísluje a zde končí číslování stránek práce;
* jednotlivé publikace se uvádějí v abecedním pořadí podle příjmení autorů a iniciál jeho jména, který se píše za čárkou;
* příjmení autora se píše velkými písmeny;
* název publikace se zvýrazňuje kurzívou;
* jestliže jsou uvedeni více než tři autoři, je možné vypsat hlavního autora s poznámkou „a kol.“(a kolektiv).

Seznam příloh

č. 1 Titulní list

č. 2 Čestné prohlášení

č. 3 Poděkování

Nepovinná část – pokud nemáte žádné přílohy ke své práci, tuto část odstraňte!

* Přílohy se zařazují na konec práce.
* Jsou to texty, obrázky, grafy, tabulky, které by přímo v textu byly zbytečně detailní, ale mají být po ruce k dokreslení východisek i výsledku řešení.
* Jsou číslovány a v textu se na ně může odkazovat.
* Před první přílohu se umisťuje seznam příloh.
* Každá příloha je označena číslem - např. Tabulka č.. 1, Schéma č. 2, Obrázek č. 3.
* Každá tabulka by měla mít i vlastní název, který stručně vystihuje její obsah.
* (Tabulka č. 1 Zakázky stavebních prací v roce 2009-2010).
* Pokud je z tabulky vytvořen graf, umístíme jej na stejné stránce jako tabulku.

**Příloha č. 1: Titulní list**