Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Programowanie gier 3D – Laboratorium

Programowanie gier 3D – lista zadań laboratoryjnych
(Data kompilacji: 28 października 2024, t: 23:54)

Spis treści

1 Wprowadzenie

Unity (http://www.unity.com), to środowisko pracy do rozwoju interaktywnych aplikacji z grafiką 3D oraz 2D. Istotną cechą środowiska Unity jest multi-platformowość. Oprócz dwóch wersji środowiska Unity dla systemów Windows oraz macOS (dostępna jest wersja dla systemu Linux), Unity pozwala na utworzenie postaci finalnej projektowanej aplikacji dla kilku różnych platform sprzętowo-programowych min. Windows, Linux, Android, IOS.

Należy dodać, iż istnieją inne darmowe systemy, lub przynajmniej oferujące możliwość darmowego tworzenia/rozwijania gier lub aplikacji 3D:

(a) Shiva Engine, https://shiva-engine.com/, **nieaktywny** ,

(b) CryEngine, http://www.crydev.net/,

(c) Project Anarchy, http://www.projectanarchy.com/, **nieaktywny** ,

(d) Torque3D, https://torque3d.org/,

(e) Neoaxis, http://www.neoaxis.com/,

(f) Evergine, https://evergine.com/,

(g) Unigine, https://unigine.com/,

(h) Unreal Engine, https://www.unrealengine.com/,

(i) Stride Engine, https://www.stride3d.net/,

(j) Open 3D Engine, https://o3de.org/,

(k) C4 Engine, http://c4engine.com/,

(l) Flax Engine, https://flaxengine.com/.

W przypadku Unity oprócz informacji oraz dokumentacji na stronach producenta:

(a) podstawowe informacje o użytkowaniu środowiska Unity – http://docs.unity3d.com/Documentation/Manual/index.html,

(b) opis komponentów, z których tworzone są aplikację w Unity – http://docs.unity3d.com/Documentation/Components/index.html,

(c) informacje o API i programowaniu w Unity – http://docs.unity3d.com/Documentation/ScriptReference/index.html.

Należy też wskazać nieco starsze strony WWW tj. Unity Community – jest to strona społeczności m.in. z dodatkowymi skryptami: https://forum.unity.com/. Przydatne informacje znajdują się także na stronie: http://www.unity3dstudent.com. Jednakże wymienione strony nie odnoszą się do najnowszych wersji środowiska Unity.

Dostępne są także serwisy, zbiory obiektów, tekstur zarówno o licencji komercyjnej, a także bezpłatnych:

(a) zbiór siatek podstawowych: https://www.thebasemesh.com/,

(b) tekstury na licencji CC0: https://ambientcg.com/,

(c) zestaw tekstur i obiektów 3D: https://polyhaven.com/,

(d) OpenGameArt: https://opengameart.org/,

(e) tekstury komercyjne, ze wsparciem dla Unity: https://quixel.com/,

(f) zestaw tekstur, o darmowej licencji ze strony GPUOpen: https://matlib.gpuopen.com/main/materials/all,

(g) obiekty komercyjne, https://3drt.com/store/, https://kitbash3d.com,

(h) Mixamo, obiekty 3D wraz z bardzo bogatym zbiorem animacji: https://www.mixamo.com,

(i) Pixel Art 2D: https://ansimuz.com/site/,

(j) zasoby Kenny NL (zasoby darmowe oraz komercyjne): https://kenney.nl/assets,

(k) zasoby ze sklepu itch.io: https://itch.io/,

(l) zasoby pozyskane z Muzeum Małopolski: https://sketchfab.com/WirtualneMuzeaMalopolski,

(m) skany obiektów historycznych: https://threedscans.com/,

Inne systemy programy wspomagające tworzenie gier:

(a) Kythera AI, https://kythera.ai,

(b) Krikey AI Animation Maker, https://www.krikey.ai/,

(c) Cascadeur – system wspomagający animację, https://cascadeur.com/,

(d) Pro Motion NG – program do tworzenia grafiki typu PixelArt, https://www.cosmigo.com/.

2 Laboratorium nr 0 – Podstawy środowiska Unity

Głównym celem tego zadania[1] jest zapoznanie się z podstawowymi elementami środowiska Unity:

- okno główne aplikacji, konfiguracja okien roboczych Unity, zasoby „Asset Store",
- rola okna Inspector, Asset, Hierarchy,
- główne komponenty dostępny w ramach systemu Unity (np.: typy świateł, komponent terenu, podstawowe bryły 3D i etc.),
- pojęcie obiektu typu „prefab" ,
- podstawowe opcje budowy końcowej postaci aplikacji.

W realizacji ćwiczenia pomocne będą materiały ze strony: https://learn.unity.com/course/unity-basics.

3 Laboratorium nr 1 – „Witaj Świecie!!!"

Przygotować aplikację w Unity, która będzie odpowiednikiem aplikacji typu „Hello World!". Napis w wersji polskiej lub angielskiej, może zostać wykreowany jako element interfejsu użytkownika wykorzystujący obiekt Canvas oraz system 2D UI. Warto wykorzystać komponent/obiekt o nazwie Text lub TextMeshPro. Druga część zadania polega na utworzeniu napisu „Hello World!" w postaci obiektu 3D oraz wykonaniu prostej animacji powstałego obiektu.

3.1 Różne uwagi związane z ćwiczeniem

- przygotować wersję binarną dla systemów typu Desktop oraz jako zadanie dodatkowe również dla systemu Android,
- napis w wersji 3D można wykonać w Unity lub dowolnym programie graficznym (np. Blender, 3ds Max) umożliwiającym eksport obiektów do Unity (polecany format FBX),
- zwrócić uwagę na kierunki układu współrzędnych w programie,
- przy eksporcie z programu graficznego warto użyć formatu FBX,
- w przypadku eksportu obiektu z pliku w formacie FBX, należy sprawdzić ile kamer oraz źródeł świateł znajduje się na scenie, ponieważ w trakcie eksportu danych mogą dołączyć dodatkowe obiekty oświetlenia bądź kamery.

4 Laboratorium nr 2 – „Tocząca się kulka"

Zaprojektować na podstawie materiału „Roll a ball" podobną prostą grę:

- https://learn.unity.com/project/roll-a-ball,

opcjonalnie w ramach rozszerzenia zadania, opracować podobny prototyp gry, składający się z trzech poziomów. Po ukończeniu przez gracza poziomu i wyświetleniu się stosownego komunikatu, ma zostać uruchomiony kolejny poziom gry.

[1] To zadanie oraz przyszłe zostały opracowane na podstawie pozycji wymienionych w literaturze oraz w oparciu o dokumentację dostępną dla środowiska Unity.

4.1 Różne uwagi związane z ćwiczeniem

Istotne elementy związane z ćwiczeniem:

- podstawy systemu GUI, kontrolka GUIText (własności: text, enabled),
- metody Start, Update, FixedUpdate, LateUpdate, OnTriggerEnter,
- klasa MonoBehaviour, funkcja Distance,
- funkcja Lerp i jej zastosowanie do zmiany pozycji obiektu, czy np.: intensywności światła,
- obsługa wejścia tj. metody typu OnMove, oraz podstawowe konfiguracji obiektu, tj. komponent Player Input,
- można również zwrócić uwagę na obiekt Input (metody: GetAxis, GetKeyUp), jest to tzw. stare (legacy) API do obsługi wejścia,
- obiekt transform (metoda Rotate, własność position), rigidbody (metoda AddForce),
- śledzenie kamerą obiektu gracza.

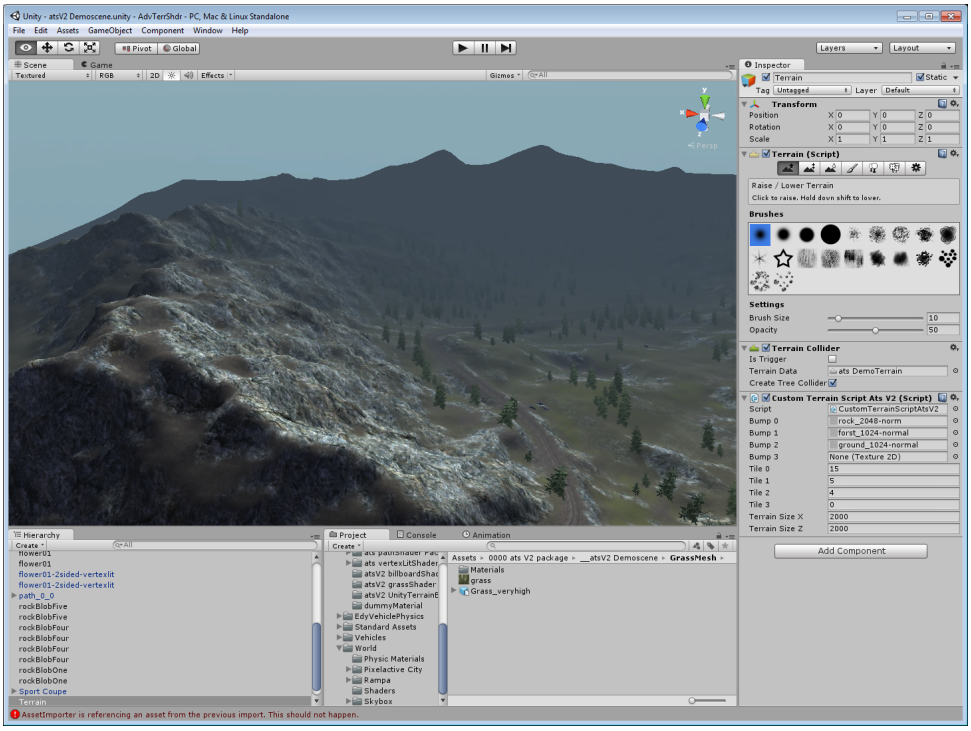
5 Laboratorium Nr 3 – Edycja terenu

Zadania jakie należy wykonać:

- utworzyć projekt zawierający obiekt/komponent terenu (lub kilka obiektów/komponentów typu teren),
- teren ma zawierać różne tekstury reprezentujące np. skały, trawę, piasek,
- dodać pojedyncze drzewa, a także grupy drzew oraz trawę przy wykorzystaniu mechanizmów Unity/Unreal,
- opcjonalnie wykorzystując „Tree Creator", dołączyć drzewa reagujące na wiatr, można też skorzystać z obiektów SpeedTree,
- *opcjonalnie ale zalecane, dodać tzw. skybox (lub alternatywnie obiekt/komponent reprezujący niebo/chmury) reprezentujący niebo nad terenem, dalsze informacje:* https://docs.unity3d.com/Manual/skyboxes-using.html,
- wykorzystując gotowy komponent Unity o nazwie FPS Character lub nowszy (zalecany) Starter Assets – First Person Character Controller[2], umożliwić graczowi poruszanie się po terenie,
- dodać element GUI (można wykorzystać tzw. Legacy API for GUI) pokazujący liczbę klatek na sekundę,
- zbudować końcową/binarną postać aplikacji (wersji binarnej naturalnie nie dołączamy do plików jakie trzeba wysłać po zakończeniu realizacji laboratorium).

Uwaga, to zadanie można wykonać w środowisku Unity, w środowisku Unreal, bądź też w środowisku Godot. W przypadku środowiska Godot, niestety nie ma domyślnego systemu terenu i należy skorzystać z dodatkowego rozszerzenie Terrain3D for Godot 4, dla wersji środowiska Godot 4.1+: https://github.com/TokisanGames/Terrain3D.

[2]Komponent FPS Character jest zawarty w Standard Asset, jednak ten pakiet jest przeznaczony dla wersji 2019 i starszych, a w kontekście wydań 2020, 2021 i 2022 lepiej wykorzystać pakiet Starter Assets – First Person Character Controller.



Rysunek 1: Przykładowy teren pochodzący z darmowego pakietu Advanced Terrain Shader V2

5.1 Uwagi związane z edycją terenu

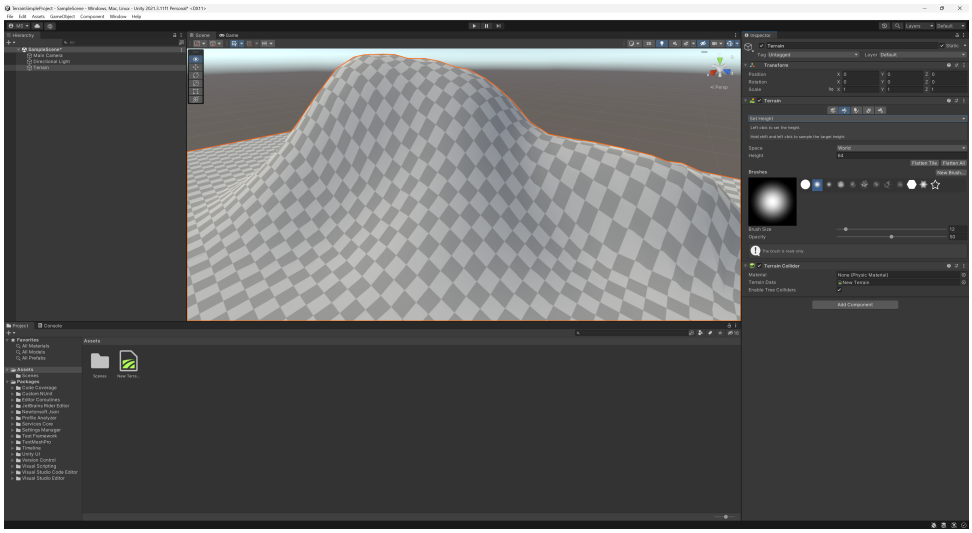
Ćwiczenie odnoszące się do komponentu teren (ang. terrain) w środowisku Unity lub systemu terenu w środowisku Unreal. Podstawowe czynności związane ze stworzeniem terenu (projekt terenu naturalnie można oprzeć o inne parametry):

- utworzyć nowy obiekt reprezentujący teren (Terrain → Create Terrain),
- ustawić wymiary X i Y terenu na 1000, natomiast wysokość na 200 (Terrain → Set Resolution),
- zastosować tzw. „Flatten Heightmap" i ustawić parametr wysokość na wartość 50 (wyjaśnić znaczenie tego operacji), Rys. (2) pokazuje umiejscowienie tej opcji w programie Unity 2021.3.11f1,
- w zakładce „Hierarchy" należy zaznaczyć „Terrain", aby w oknie Inspector pojawiły się opcje edycji terenu.

Podstawowe funkcje edycji terenu znajdują się w oknie Inspektor na Rys. (1):

- na Rys. (1) zaznaczona została funkcja do podnoszenia lub obniżania terenu. Można zastosować kilka rodzajów pędzla także można ustawić rozmiar pędzla jak również kształt. Uwaga, aby obniżyć wartość wysokości terenu należy „malować" mając wciśnięty klawisz Shift,
- inne możliwości edytora dotyczące samego kształtu terenu to m.in. wyrównywanie terenu do danej wysokości i wygładzanie,
- podczas edycji terenu pomocna jest możliwość zmiany widoku, za pomocą prawego i środkowego klawisza myszki (obrót, przybliżenie, przesunięcie). Alternatywnie można skorzystać z róży kierunków widocznej w prawym górnym rogu sceny.

Dodatkowe tekstury reprezentujące teren można zainstalować z darmowych pakietów „Terrain Sample Asset Pack". Tekstura wody dostępna jest w pakiecie „Simple Water Shader URP" wymaga jednak to zastosowania potoku renderingu URP. Dla podstawowego potoku można zastosować pakiet „AQUAS Lite - Built-In Render



Rysunek 2: Opcja „flatten heightmap” – poziomowanie mapy wysokości

Pipeline”. Pakiet „Unity Terrain - HDRP Demo Scene” zawiera tekstury oraz obiekty o bardzo wysokiej jakości, co zwiększa wymagania co do stosowanej karty graficznej.

Aby rozpocząć proces nakładania tekstur należy w oknie inspektora wybrać opcję malowania tekstur (ikona „Pędzla” – Paint texture). Klikając w przycisk „Edit Terrain Layers...”, można dodać tzw. warstwę. Warstwy stanowią zestaw tekstur do nakładania na teren. Pierwsza warstwa stanowi swoisty podkład i pokrywa całość terenu.

Opcjonalnie ale zalecane. Chcąc dodać element typu SkyBox, w pierwszej kolejności należy zaimportować pliki reprezentujące SkyBox, i ustawić wartość Wrap Mode na Clamp. Następnie należy utworzyć nowy materiał (Asset → Create → Material), nazwać go mp.: cubemap i zmienić typ shadera na SkyBox (RenderFx → SkyBox). Następnie należy dodać poprawnie wszystkie tekstury tak, aby poprawnie odzwierciedlały niebo.

Elementem poprawiającym jakość grafiki będzie tzw. efekt flary. Należy zaimportować pakiet Light Flares. Następnie dodać do sceny światło kierunkowe (GameObject → Create Other → Directional Light). W oknie Insepctor'a niezbędna jest następująca zmiana Flare → Sun. Odpowiednio należy również zmienić pozycję oraz obrót źródła światła,by oświetlało modelowany teren oraz aby źródło światła znajdowało się w tym samym miejscu, co słońce na teksturach tworzących SkyCube.

Obsługę poruszania się gracza można oprzeć o pakiet „First Person Character Controller”. Należy jednak zwrócić uwagę na kamerę, należy się upewnić, że jest obecna tylko jedna kamera. Należy, też sprawdzić parametry obiektu reprezentującego gracza, aby poprawnie dobrać wartości skali, rozmiaru, czy też sposobu poruszania się.

6 Laboratorium nr 4 – Realizacja gry typu Pong

Głównym celem tego zadania jest zrealizowanie za pomocą środowiska Unity, aplikacji odnoszącej się do gry Pong (dodatkowe informacje http://en.wikipedia.org/wiki/Pong). Ogólny koncept gry, która ma być zrealizowana jako ćwiczenie przedstawia Rys. 3.

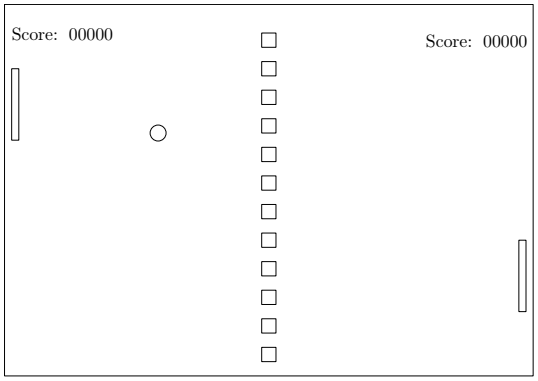
W projekcie gry można wykorzystać rzut perspektywiczny lub rzut równoległy, odpowiednio dobierając parametry kamery głównej. Gra ma obejmować zliczanie punktów, gdy odbita piłka uderzy w ścianę za paletką.

W grze opcjonalnie może zostać zaimplementowane menu, operacja przerwania gry, opuszczenie (zamknięcie) aplikacji. Podczas zamykania aplikacji ma zostać wyświetlony dodatkowy ekran np.: informacją o autorze. Jednakże, zadanie dot. menu i dodatkowe akcje są treścią laboratorium nr 7.

6.1 Uwagi związane z grą typu Pong

Utworzenie elementów planszy można oprzeć o obiekcie „Cube”, natomiast punktacja może zostać wyświetlona za pomocą komponentu „GUI Text”.

Poruszanie się kulki, a także obsługę kolizji można zrealizować na dwa sposoby:



Rysunek 3: Ogólny koncept grafiki w grze Pong, umieszczone na środku kwadraty pełnią rolę dekoracyjną

- samodzielnie implementując ruch kulki oraz wykrywając zderzenie, obliczać wartość wektora odbicia za pomocą metody Reflect z obiektu Vector3,
- wykorzystać system symulacji fizyki dostępny w Unity.

Ogólnie aplikacja, Pong posiada cztery obszary na jakie trzeba zwrócić uwagę:

- sterowanie zachowaniem się „kulki", czyli właściwe odbicie, „podkręcanie" kulki przez paletkę gracza,
- obszar gry, czy kulka może opuścić obszar gry, czy odbija się od brzegu,
- sterowanie paletką gracza,
- sterowanie paletką przez komputer,
- przechowywanie informacji pomiędzy ładowanymi scenami można zrealizować wykorzystując metodę „DontDestroyOnLoad", więcej informacji znajduje się pod adresem: http://docs.unity3d.com/Documentation/ScriptReference/Object.DontDestroyOnLoad.html.

Przykładowy skrypt sterujący piłką (należy zwrócić uwagę na poprawność kodu skryptu):

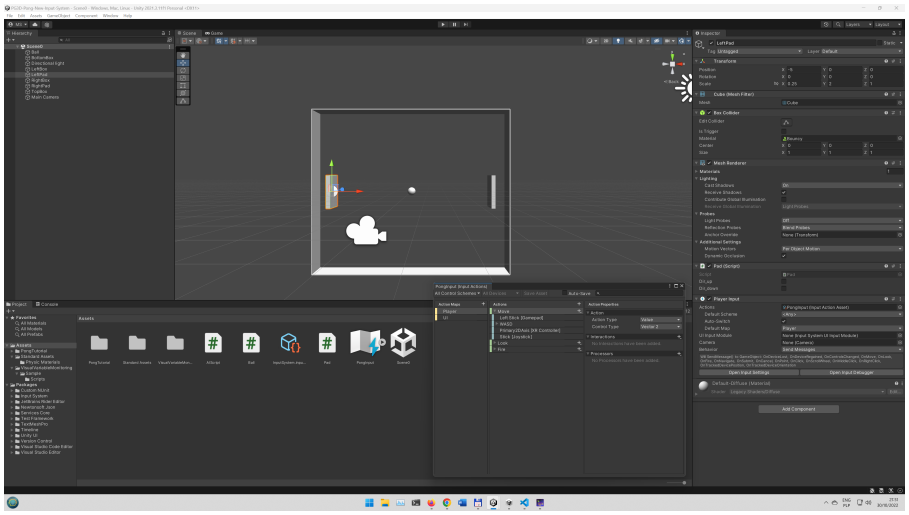
```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Ball : MonoBehaviour {
5
6
7          public float fltSpeed = 5.0f;
8          public float fltTimeFactor = 10.0f;
9
10         void Start () {
11                 rigidbody.AddForce( 5.0f, 5.0f, 0.0f);
12         }
13
14         void Update () {
15                 var fromVel = rigidbody.velocity;
16                 var toVel = fromVel.normalized * fltSpeed;
17                 rigidbody.velocity = Vector3.Lerp(fromVel, toVel, Time.deltaTime
18                                                                 * fltTimeFactor);
19         }
20
21         void OnCollisionEnter(Collision col) {
22                 if(col.gameObject.name == "LeftPad") {
23                         if ( col.gameObject.GetComponent<Pad>().dir_up == true) {
24                                 rigidbody.AddForce(0,-100,0);
25                         }
26                         if ( col.gameObject.GetComponent<Pad>().dir_down == true) {
27                                 rigidbody.AddForce(0, 100,0);
28                         }
29                 }
30         }
31 }
```

Skrypt sterujący paletką:

```
using UnityEngine;
using System.Collections;

public class Pad : MonoBehaviour {

        public KeyCode key_up;
        public KeyCode key_down;

        public bool dir_up;
        public bool dir_down;

        void Start () {
                dir_up = false;
                dir_down = false;
        }

        void Update () {

        }

        void FixedUpdate() {

                if (Input.GetKey(key_up)) {
                        transform.Translate(new Vector3(0,5.0f,0) * Time.deltaTime);

                        dir_up = true;
                }
                else {
                        dir_up = false;
                }

                if (Input.GetKey(key_down)) {
                        transform.Translate(new Vector3(0,-5.0f,0) * Time.deltaTime);

                        dir_down = true;
                }
                else {
                        dir_down = false;
                }
        }

}
```

Implementację sterowania paletką przez komputer można zrealizować odczytując współrzędne piłki i przesuwając paletkę zgodnie z kierunkiem zmian na osi Y:

```
using UnityEngine;
using System.Collections;

public class AIScript : MonoBehaviour {

        public float moveSpeed = 5.0f;

        private GameObject ball = null;

        void Start () {

                ball = GameObject.Find("Ball");

        }

        void LateUpdate () {
                if(ball.transform.position.y > transform.position.y) {
                        transform.Translate(new Vector3(0,moveSpeed,0) * Time.deltaTime);
                }
                if(ball.transform.position.y < transform.position.y) {
                        transform.Translate(new Vector3(0,-moveSpeed,0) * Time.deltaTime);
                }
        }
}
```



Rysunek 4: Podstawa konfiguracja lewej paletki tj. paletki gracza dla nowego systemu Input, w przykładzie wykorzystuje się nowy system wejścia

Przydatne informacje dotyczące czynności związanych z zamykaniem aplikacji znajdują się pod adresem:

- http://docs.unity3d.com/Documentation/ScriptReference/Application.CancelQuit.html

6.2 Wykorzystanie nowego systemu Input

W projekcie „Pong" należy też wykorzystać nowy system Input (choć zaleca się realizację dwóch projektów stosując stary oraz nowy system Input). Instalację nowego systemu należy rozpocząć od instalacji pakietu Input System za pomocą okna Package Manager. Najważniejsze elementy konfiguracji przedstawia Rys. 4.

Założono, iż sterowanie odbywać się będzie za pomocą domyślnego systemu akcji oraz metody OnMove. Paletka sterowana przez gracza wykonuje ruch tylko np. po osi Y, zatem w metodzie OnMove korzystamy tylko z wartości Y. Kod źródłowy klasy Pad po wykonaniu niezbędnych zmian przedstawia się następująco:

```
1  using UnityEngine;
2  using System.Collections;
3  using UnityEngine.UI;
4  using UnityEngine.InputSystem;
5
6
7  public class Pad : MonoBehaviour {
8
9          public bool dir_up;
10         public bool dir_down;
11
12         float movementX;
13         float movementY;
14
15         void Start () {
16                 dir_up = false;
17                 dir_down = false;
18
19                 movementX = 0;
20                 movementY = 0;
21         }
22
23         void Update () { }
24
25         void FixedUpdate() {
26
27                 if (movementY > 0.0f) {
28                         transform.Translate(new Vector3(0,5.0f,0) * Time.deltaTime);
29
30                         dir_up = true;
31                 }
```

```
32                  else {
33                          dir_up = false;
34                  }
35
36                  if (movementY < 0.0f) {
37                          transform.Translate(new Vector3(0,-5.0f,0) * Time.deltaTime);
38
39                          dir_down = true;
40                  }
41                  else {
42                          dir_down = false;
43                  }
44
45          }
46
47          private void OnMove(InputValue movementValue) {
48
49                  Vector2 movementVector = movementValue.Get<Vector2>();
50
51                  movementX = 0;
52                  movementY = movementVector.y;
53          }
54  }
```

7 Laboratorium nr 5 – System menu w grze

Głównym celem tego zadania będzie realizacja systemu menu jaki można napotkać w większości gier komputerowych. Menu można opracować w systemie GUI oferowanym w Unity (przydatny będzie darmowy pakiet z Asset Store – https://assetstore.unity.com/packages/essentials/ui-samples-25468).

Komentarz 7.1 *Istnieje też możliwość skorzystania z darmowego pakietu NGUI, jednakże należy w tym przypadku korzystać ze starszych wersji Unity.*

Pakiet NGUI 2.7 można uzyskać pod adresem:

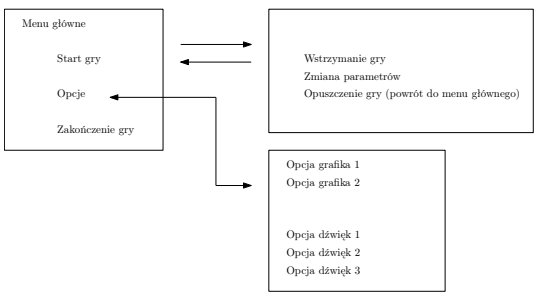
- *http://www.tasharen.com/?page_id=140*

Dodatkowe informacje o pakiecie NGUI znajdują się pod adresem: http://www.tasharen.com/forum/index.php?topic=6754. Niestety, pakiet NGUI w wersji 2.7 jest dedykowany starszym wydaniom Unity (wydanie 2.7 pochodzi z 2013 roku). Nowe wersje NGUI są oferowane na zasadach komercyjnych.

Dlatego, do rozwiązania zadania zalecane jest wykorzystanie tzw. nowego systemu GUI jaki obecny jest w aktualnym wydaniu środowiska Unity. Wykorzystanie starego systemu („legacy") jest opcjonalne.

W menu ma znaleźć się menu główne, menu z opcjami np.: dotyczącymi grafiki czy muzyki. A także menu dostępne w trakcie gry, wywoływane za pomocą klawisza ESC, pozwalające m.in. na zatrzymanie, gry, opuszczenie gry, zamknięcie aplikacji. Należy przygotować menu dwujęzyczne w języku polskim oraz angielskim (z możliwością dodania później innych języków). Zakładamy, iż menu wywoływane klawiszem ESC będzie oknem, które można przesuwać np.: myszką.

Ogólny i bardzo poglądowy schemat menu przedstawia Rys. 5.



Rysunek 5: Ogólny schemat projektowanego menu w ramach zadania

7.1 Uwagi związane z zadaniem

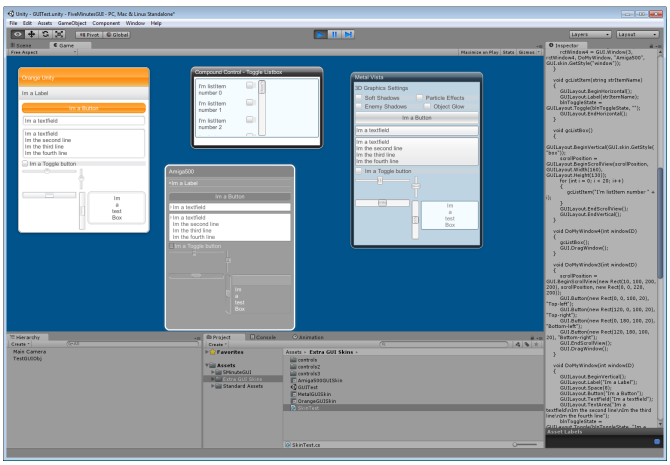
Przydatne informacje o realizacji menu dla starszej i nowszej wersji GUI można odszukać w dokumentacji Unity. Ogólnie główne komponenty jakie mogą okazać się przydatne w realizacji zadania to:

- klasa GUI,
- okna DragWindow, ModalWindow, Window,
- komponenty Box, Button, Label, Toggle,
- TextArea, TextField, PasswordField,
- BeginScrollView, EndScrollView,
- BeginGroup, EndGroup,
- Toolbar, DrawTexture, FocusControl FucusWindow.

Główną metodą do obsługi tzw. starego/legacy GUI, w klasach dla obiektów typu GameObject jest OnGUI, w tej metodzie powinna być realizowana obsługa interfejsu użytkownika. Klasa GUI zawiera elementy interfejsu użytkownika, które nie są automatycznie rozmieszczane, zawsze należy podać współrzędne pod którymi dany komponent ma się znaleźć. Inaczej funkcjonuje llasa „GUILayout", zawiera podobny zestaw komponentów co GUI, ale elementy są rozmieszczane automatycznie, można np.: określić iż kolejne przyciski mają być umieszczone pionowo lub w poziomie np.:

```
1 GUILayout.BeginVertical("box");
2 GUILayout.Button("I'm the first button.");
3 GUILayout.Button("I'm the second button.");
4 GUILayout.EndVertical();
```

System GUI w Unity posiada także wsparcie dla stylów, darmowe style jakie można zastosować to „Extra GUI Skins" lub „Built-in GUI Skin" obydwa dostępne w Asset Store.



Rysunek 6: Okna uzyskane za pomocą stylów dostępnych w pakiecie „Extra GUI Skins"

8 Laboratorium nr 6 – Skanowanie obiektów 3D

Wykorzystując dostępny w laboratorium skaner 3D wykonać skan wybranego obiektu. Dodatkowa obróbka, aby uzyskać obiekt o niskiej rozdzielności może być ograniczona do podstawowej operacji decymacji siatki trójkątów np. w programie Blender. Dostępny skaner 3D w laboratorium również pozwala na utworzenie obiektu 3D i jego podstawową obróbkę. Więcej, i bardziej zaawansowane algorytmy do przetwarzania obiektu oraz tekstur oferuje program MeshLab. Obiekt pozyskany poprzez procedurę skanowania powinien posiadać teksturę.

8.1 Uwagi związane z zadaniem

Dodatkowe (nie jest to obowiązkowe) zadanie, polega na porównania jakości otrzymanego obiektu przy zastosowaniu dedykowanego oprogramowania skanera z możliwościami programu MeshLAB. Dostępny w pracowni skaner wykonuje również zdjęcia skanowanego obiektu. Pozyskane zdjęcia można wykorzystać do realizacji procedury fotogrametrii, aby sprawdzić, czy możliwe jest wykorzystanie ich do rekonstrukcji obiektu 3D.

Oprogramowanie do fotogrametrii np. VisualSFM, CMSV, MeshLAB, Meshroom również pozwoli na wykonanie rekonstrukcji wybranego obiektu. Dodatkowe w tym momencie zadanie, może polegać na porównaniu jakości otrzymanego obiektu przy zastosowaniu podanych programów oraz dedykowanego oprogramowania jakie jest obecnie dostępne do realizacji rekonstrukcji obiektu. Dostępny w pracowni skaner wykonuje również zdjęcia skanowanego obiektu. Pozyskane zdjęcia można wykorzystać do realizacji procedury fotogrametrii, aby sprawdzić, czy możliwe jest wykorzystanie ich do rekonstrukcji obiektu 3D.

Dalsze dodatkowe informacje dotyczące technik fotogrametrii można odszukać m.in. za pomocą poniższych linków:

- https://github.com/mikeroyal/Photogrammetry-Guide,
- https://github.com/quilime/photogrammetry,
- https://github.com/awesome-photogrammetry/awesome-photogrammetry,
- https://github.com/openMVG/awesome_3DReconstruction_list,
- https://github.com/natowi/photogrammetry_datasets,
- https://github.com/natowi,
- https://openheritage3d.org.

Komentarz 8.1 *Oprogramowanie komercyjne tj. Agisoft Metashape (wersje Professional, Standard), 3DF Zephyr oraz Reality Capture (jest dostępna wersja darmowa od wydania 1.4.x) oferują bezpłatne wersje demo bądź trial. Zatem też można je wykorzystać do realizacji zadań w oparciu o technikę fotogrametrii.*

9 Laboratorium nr 7 – Animacja obiektów 3D

Unity oferuje system Mecanim do sterowania procesem animacji. Zadanie postawione w tym laboratorium składa się z trzech głównych zagadnień:

(I) proste animacje, z wykorzystaniem dostępnego systemu Mecanim, np. otwarcie drzwi, obrót obiektu, przesunięcie obiektu,

(II) wykorzystanie pakietu iTween[3],

(III) wykonanie animacji postaci, tj. bieganie, chodzenie w ramach systemu Unity.

9.1 Uwagi związane z zadaniem

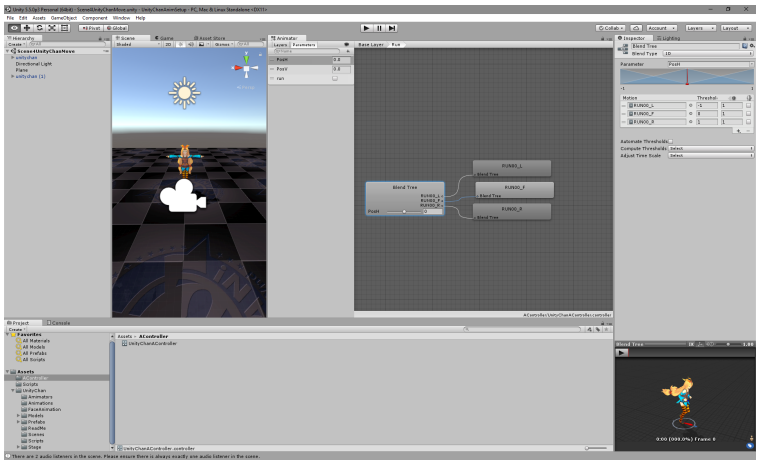
Projekt odnoszący się do laboratorium naturalnie ma być jeden, ale każde zagadnienie ma być zrealizowane w oddzielnej scenie. Warto dodać dodatkową scenę z menu, które będzie pozwalać na wybór jednej z trzech scen. Sceny (I) oraz (II) mogą przedstawiać podobne przykłady, ale sposób ich animacji należy zaprezentować z pomocą podstawowych mechanizmów Mecanim oraz iTween.

W przykładzie (III) można skorzystać z gotowej postaci Unity Chan[4], postać posiada 31 różnych schematów animacji oraz także mimikę ruchu ust, można rozważyć użycie postaci dostępnych z systemu Mixamo[5] lub pakietu Cascadeur[6] albo Blender[7].

W procesie tworzenia sterownika animacji, należy pamiętać o korzystaniu z parametrów jakie można wprowadzić do sterownika, co w efekcie uprości sterowania animacją postaci:

[3] Adres: http://www.pixelplacement.com/itween/index.php
[4] Adres: https://assetstore.unity.com/packages/3d/characters/unity-chan-model-18705
[5] Adres: https://www.mixamo.com/
[6] Adres: https://cascadeur.com
[7] Adres: https://www.blender.org/

```
Animator animator;

void Start () { animator = GetComponent<Animator>(); }

void Update () {
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");
    bool fire = Input.GetButtonDown("Fire1");

    animator.SetFloat("Forward",v);
    animator.SetFloat("Strafe",h);
    animator.SetBool("Fire", fire);
}

void OnCollisionEnter(Collision col) {
    if (col.gameObject.CompareTag("Enemy")) {
        animator.SetTrigger("Die");
    }
}
```



Rysunek 7: W przykładzie animacji (III) do uzyskania płynnego przejścia np. od biegu do marszu należy zastosować drzewa nakładania animacji

10 Laboratorium nr 8 – Kamera typu FPS w środowisku 3D

Zadanie do zrealizowania w ramach laboratorium, polega na stworzeniu prototypu gry 3D typu FPS (ang. first-person camera[8]). Gracz ma poruszać się swobodnie w środowisku 3D, w tym celu można wykorzystać pakiet „Character Controller".

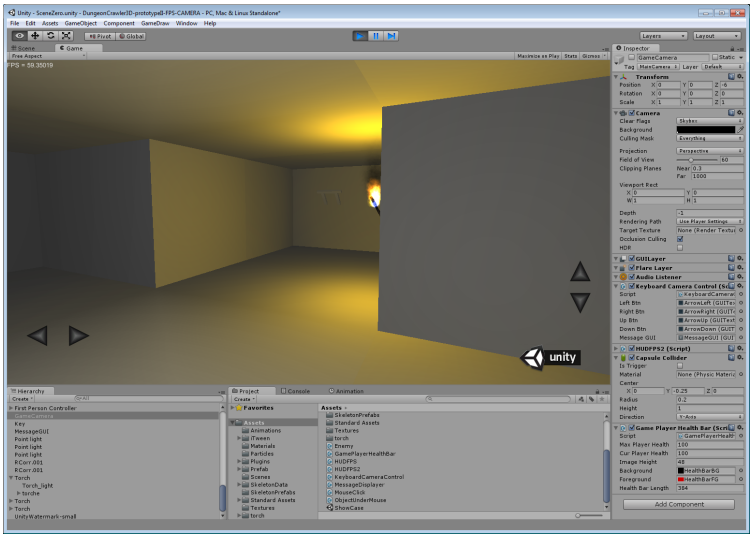
Poziom gry może zostać zaprojektowany z elementów przygotowanych np.: w programie Blender. Należy zwrócić uwagę na zwrot wektorów normalnych. A także zaprojektować elementy ruchome jak przełącznik do windy, otwierane drzwi bezpośrednio, a także drzwi wymagające dodatkowego klucza. Obiekt pochodni pochodzi ze Asset Store, jest to obiekt o nazwie „Simple Torch", adres https://assetstore.unity.com/packages/3d/props/interior/simple-torch-7275.

Prototyp gry ma zwierać poziom, którego konstrukcja jest zróżnicowana w pionie, może to być budynek z windą, przy czym obecność windy jest obowiązkowa.

10.1 Uwagi związane z zadaniem

Wykorzystanie skryptów typu „Character Controller"[9], umożliwia łatwe zrealizowanie poruszania się w grze. Zmianę kierunku „spojrzenia" kamery za pomocą myszy należy zmienić, aby była realizowana tylko wtedy, gdy

[8] Realizacja pełnego kontrolera FPS jest zadaniem laboratorium nr 10, punkt 12.
[9] Przykładem może być pakiet: Starter Assets - First Person Character Controller.



Rysunek 8: Projektowany poziom w grze, obiekt pochodni dostępny w Asset Store

użytkownik np.: naciśnie lewy przycisk myszy. Istotnym zadaniem jest wykrywanie, jaki obiekt został wskazany przez użytkownika. Należy w tym celu wykorzystać metodę ScreenPointToRay, która konwertuje wskazane współrzędne 2D, z uwzględnieniem rzutu kamery, najprostszy schemat wykrywania jaki element został wskazany jest następujący:

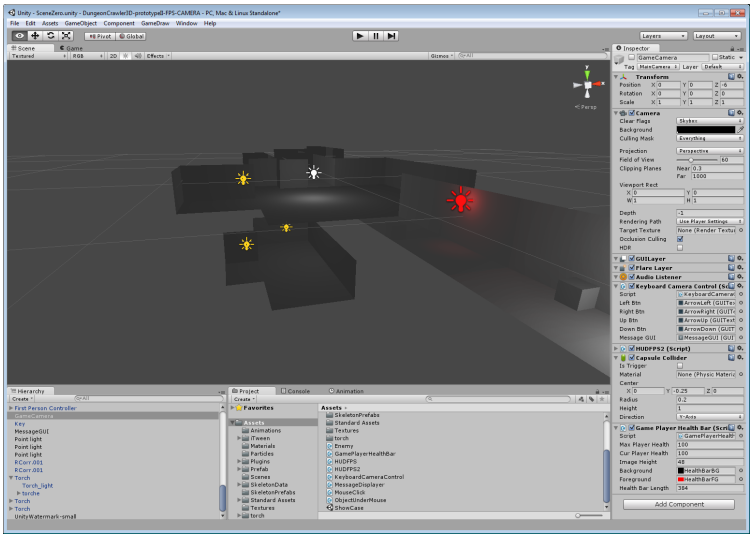
```
if ( Input.GetMouseButtonDown(0) )
{
        ray = Camera.main.ScreenPointToRay (Input.mousePosition);

        if (Physics.Raycast (ray, out hit, 100.0f))      {
                if( hit.collider.name == "Door") {
                        print("animate-door");
                        Door.animation.Play("OpenDoor");
                }
                // PL: testy dla innych obiektów
                // EN: tests for other objects
        }
}
```

W prototypie ma pojawić się podświetlanie obiektów np.: zmiana koloru klucza, jeśli użytkownik przesuwa kursor myszy nad obiektem. Można to zrealizować za pomocą oddzielnego skryptu, umieszczając w nim dwie metody: OnMouseEnter, która zostanie wywołana, gdy mysz znajdzie się nad obiektem oraz OnMouseLeave wywołana w momencie, gdy kursor myszy opuści obszar obiektu:

```
using UnityEngine;
using System.Collections;

public class ObjectUnderMouse : MonoBehaviour {

        void OnMouseEnter()      {
        renderer.material.color = new Color(1.0f, 0.0f, 0.0f);
        }

        void OnMouseExit() {
        renderer.material.color = new Color(0.91f, 0.91f, 0.09f);
        }
}
```

Przesunięcie np.: obiektu windy, np.: pomiędzy piętrami można rozpocząć od wykrycia czy użytkownik wskazał odpowiedni obiekt:

```
if( hit.collider.name == "ElevatorSwitch" && ElevatorSwitch.animation.isPlaying == false
        && ElevatorSwitchActive == false) {
        ElevatorSwitchActive=true;
```

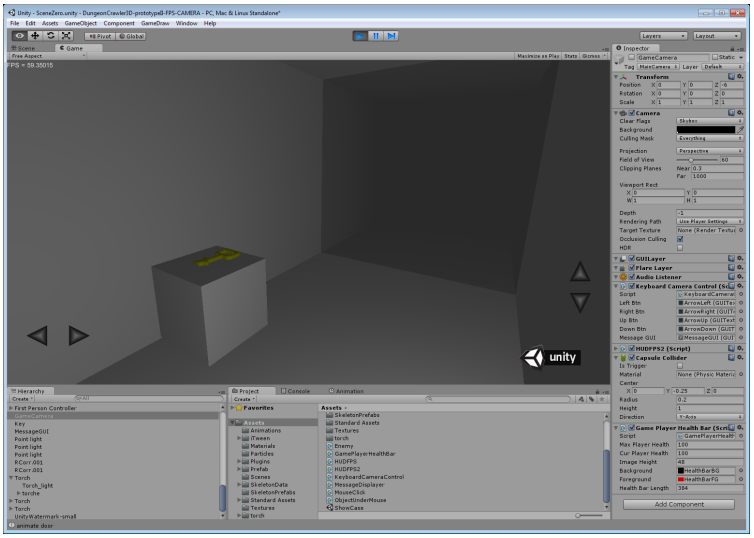


Rysunek 9: Ogólne spojrzenie na projektowany poziom w grze, wektory normalne powinny być skierowane do wewnątrz

```
4           print("animate switch 1");
5           ElevatorSwitch.animation.Play("ActiveSwitch");
6
7           ElevatorIsMoving=true;
8           Invoke("ElevatorMoveDown", 1.1f);
9   }
10
11  if( hit.collider.name == "ElevatorSwitch" && ElevatorSwitch.animation.isPlaying == false
12          && ElevatorSwitchActive == true) {
13          ElevatorSwitchActive=false;
14          print("animate switch 2");
15          ElevatorSwitch.animation.Play("ActiveSwitch2");
16
17          ElevatorIsMoving=true;
18          Invoke("ElevatorMoveUp", 1.1f);
19  }
```

Zależy to także od aktualnej pozycji windy, czy znajduje się na „piętrze" czy też na „parterze". Przesunięcie windy jest realizowane za pomocą pakietu iTween https://assetstore.unity.com/packages/tools/animation/itween-84, po zakończeniu procesu przesunięcia odpowiednich obiektów, wywołana zostanie metoda MoveComplete.

```
1   private void ElevatorMoveUp() {
2           iTween.MoveBy (Elevator, iTween.Hash("z", 4.0f, "time", 8, "delay", 0.0, "looptype",
3                   iTween.LoopType.none) );
4           iTween.MoveBy (gameObject, iTween.Hash("y", 4.0f, "time", 8, "delay", 0.0, "looptype",
5                   iTween.LoopType.none, "oncomplete", "MoveComplete") );
6   }
7
8
9   private void ElevatorMoveDown() {
10          iTween.MoveBy (Elevator, iTween.Hash("z", -4.0f, "time", 8, "delay", 0.0, "looptype",
11                  iTween.LoopType.none) );
12          iTween.MoveBy (gameObject, iTween.Hash("y", -4.0f, "time", 8, "delay", 0.0,
13                  "looptype", iTween.LoopType.none, "oncomplete", "MoveComplete") );
14  }
15
16  private void MoveComplete() {
17          ElevatorIsMoving = false;
18          print("elevator moving end");
19  }
```



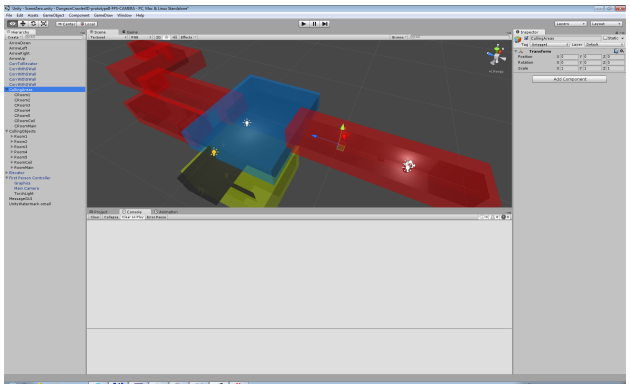
Rysunek 10: W prototypie gry powinny znaleźć się także drzwi oraz obiekty, które można zbierać np.: klucz

10.2 Optymalizacja sceny graficznej

Algorytmy usuwania niewidocznych (przesłoniętych przez inne obiekty 3D) elementów sceny są istotnym elementem optymalizacji aplikacji graficznych. Pozwala to na zwiększenie końcowej wydajności aplikacji, obniżenie potencjalnych wymagań. Optymalizacja, pozwala także na budowę bardziej złożonej grafiki, bowiem moc obliczeniowa nie jest tracona na obsługę obiektów, które nie są wyświetlane, nie są przetwarzane przez podsystem grafiki danej aplikacji. Jednakże, należy pamiętac iż brak widoczności obiektu może jednak nadal wymagać realizacji skryptów i obsługi fizyki (lub logiki gry).

Unity oferuje zintegrowany system, który realizuje tego typu optymalizację. Rozwiązanie to jest obecnie dostępne w każdej wersji Unity. Warto także sprawdzić pakiet o nazwie SECTR VIS dostępny w Asset Store[10].

Głównym zadaniem tego laboratorium jest dołączenie do projektu, w oparciu o poprzedni przykład prototypu gry z labiryntem, systemu optymalizującego proces wizualizacji labiryntu. W zadaniu tym należy zastosować system Umbra dostępny w środowisku Unity. Jest to okno o nazwie „Occlusing Culling" dostępne z menu Window.



Rysunek 11: Przykładowy podział elementów/obiektów na obszary, które będą ukrywane dynamicznie w zależności od pozycji gracza. Przykład dotyczy pakietu M2H, który pozwala określać rożne kolory dla poszczególnych obiektów otaczających. W zadaniu jednak należy korzystać z rozwiązania oferowanego przez Unity.

[10]Istnieje też darmowe rozwiązanie o nazwie M2H Culling (jednakże obecnie pakiet ten nie jest bezpośrednio dostępny), dostępne w Asset Store, dodatkowe informacje można odszukać pod adresem: http://www.m2h.nl/unity/. Autorem systemu M2H Culling jest Mike Hergaarden.

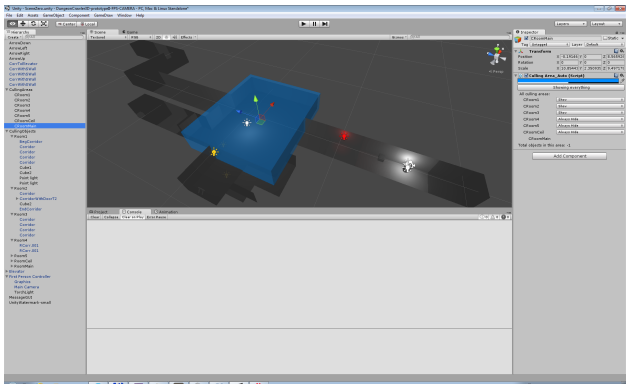
10.3 Uwagi związane z zadaniem

Stosowanie pakietu M2H nie jest zalecane, należy zastosować podejście oparte o system Umbra dostępny w środowisku Unity. Uruchominie procesu optymalizacji wykonujemy poprzez okno[11] „Occlusing Culling". Ogólnie tworząc projekt do zadnia, należy zwrócić uwagę na prawidłowe określenie obszaru okluzji tzw. Occlusion Area[12]. Ważnym pojęciem jest również tzw. portal, który łączy dwa obszary ze sobą[13].

Bez względu na stosowane rozwiązanie, zalecane jest utworzenie poniższej hierarchii (lub podobnej). W pierwszym obiekcie CullingObject, umieszczone są poszczególne elementy sceny podzielone na dodatkowe obszary, tworząc całą hierarchię opisującą pomieszczenia stanowiące określony poziom gry np.:

- CullingObject
  - PokójStartowy
  - PokójPrzejściowy1
  - PokójPrzejściowy2
  - PokójPomocniczy
  - Magazyn
  - PokójKońcowy

Należy zwrócić uwagę na to, aby wszystkie obiekty sceny znajdowały się w danym pomieszczeniu.



Rysunek 12: Ustalenie obszaru otaczającego dla głównego pomieszczenia

10.4 Dalsze informacje o starszym pakiecie M2H

Komentarz 10.1 *Stosowanie pakietu M2H nie jest zalecane ani wymagane, należy zastosować pakiet Umbra dostępny w Unity. Dalsze informacje pochodzą ze starszych wersji listy zadaniowej. Podane poniżej informacje mają tylko charakter informacyjny.*

Skrypty z pakietu M2H Culling zakładają, iż obecne będą obiekty CullingAreas oraz CullingObjects o tak dokładnie określonych nazwach. Choć dzięki obecności kodu źródłowego można zmienić podane nazwy na inne. Należy dodać, iż w drugim obiekcie CullingAreas znajdują się obiekty otaczające obszary stanowiące spójne obszary gry.

Istnieją dwa systemy proponowane przez M2H Culling, „manual" i „auto". Podsystem „manual", pozwala precyzyjnie określać jakie obiekty będą ukrywane. System „auto", samodzielnie wyszukuje obiekty, wg. utworzonej hierarchii. W ćwiczeniu dla uproszczenia, a także przyspieszenia pracy, warto zastosować system „auto".

[11] Podstawowe informacje o tej technologii zawarto w: https://docs.unity3d.com/2020.1/Documentation/Manual/OcclusionCulling.html

[12] Informacje o tym pojęciu znajdują się pod adresem: https://docs.unity3d.com/2020.1/Documentation/Manual/class-OcclusionArea.html

[13] Rola tego pojęcia jest przedstawione w: https://docs.unity3d.com/2020.1/Documentation/Manual/class-OcclusionPortal.html

W trakcie zadania wykorzystuje się tylko dwa skrypty zawierający klasy: CullingArea_Auto, który należy podłączyć do obiektu otaczającego oraz CullingCamera_Auto zgodnie z nazwą należy podłączyć do kamery gracza. Potrzebna jest też klasa CullingAreaEditor_Auto współpracująca z edytorem Unity w trakcie projektowania poziomu gry.

Należy utworzyć dwie hierarchie obiektów oraz obszarów ukrywających, a następnie dla obiektów ukrywających należy dodać skrypt CullingArea_Auto i przesunąć obiekt oraz go przeskalować tak aby objął zestaw obiektów 3D, które mają być ukrywane. Dodatkowo, należy w każdym obiekcie ukrywającym ustalić jakie inne obiekty ukrywające mają być zawsze pokazywane bądź ukrywane.

Dodatkowe informacje można odszukać w dokumencie z pod adresu: http://www.m2h.nl/files/Unity_occlusion_system_by_M2H.pdf.

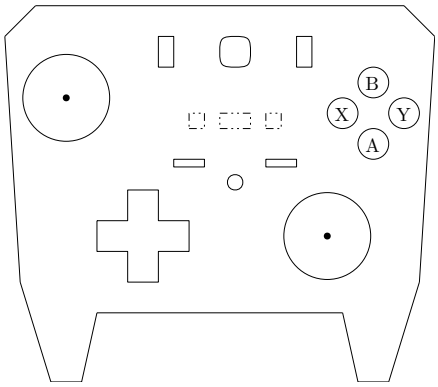
11 Laboratorium nr 9 – Urządzenia wejścia

Zadanie do zrealizowania polega na odczytaniu informacji z trzech dostępnych w laboratorium kontrolerów:

- gamepada,
- joysticka,
- kierownicy.

Projekcie powinno się znajdować menu główne, w którym będzie można wybrać z jakiego typu kontrolera będą odczytywane informacje. Po wyborze kontrolera, powinien pokazać się interfejs użytkownika, gdzie można będzie odczytać oraz zobaczyć wizualizację odczytywanych danych.

W zależności od typu wybranego urządzenia na scenie należy pokazać schemat danego urządzenia ukazujący jak zachowują się poszczególne elementy sterujące. Rysunek 13 pokazuje przykładowy schemat dla gamepada.



Rysunek 13: Schemat gamepada jaki powinien pokazywać się podczas odczytu informacji z urządzenia

11.1 Uwagi związane z zadaniem

Odczyt informacji należy zrealizować za pomocą tzw. nowego Input System. Przydatne będą także następujące informacje z dokumentacji:

- informacje dotyczące obsługi gamepada: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/Gamepad.html
- informacje dla joysticka: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/Joystick.html,
- oraz ogolnie informacje dla urządzeń HID pomocne dla tworzenia własnego układu (custom layout): https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/HID.html, przydatne podjeście w kontekście rozbudowawnego joysticka lotniczego bądź kierownicy.

12 Laboratorium nr 10 – Kontroler FPS

Zadanie polega na opracowaniu własnego zestawu komponentów (bądź jednego komponentu jeśli okaże się to możliwe), w postaci skryptów realizujących poruszanie się postaci gracza w trybie kamery pierwszoosobowej. Opracowany zestaw komponentów powinien obejmować następujące zachowania gracza:

- zwykły tryb chodzenia (poruszanie się postaci do przodu/tyłu oraz na boki),
- rozglądanie się za pomocą myszy,
- tryb biegania aktywowany za pomocą dodatkowego klawisza,
- tryb kucania, również aktywowany innym klawiszem,
- skakanie,
- równoważenie prędkości poruszania się po powierzchni płaskiej i nachylonej,
- "ślizganie się".

W realizacji zadania dodatkowe założenia jakie można przyjąć to:

- obsługa tradycyjnej pary kontrolerów tj. klawiatura plus mysz,
- obsługa gamepada,
- lepsze wykorzystanie nowego systemu Input,
- wprowadzenie obsługi ograniczonego trwania np. biegu, czyli mechaniki wytrzymałości postaci.

Do zadań opcjonalnych przynależą m.in. następujące zagadnienia:

- rzucanie przedmiotami,
- skakanie, poruszanie się po ścianach,
- wspinanie się po powierzchniach pionowych,
- obsługa doskoku,
- poruszanie się za pomocą liny,
- obsługa broni tj. strzelanie, po trajektorii prostej oraz ukośnej,

12.1 Uwagi związane z zadaniem

Sterowanie graczem zasadniczo realizuje się za pomocą systemu fizyki, np. po ustaleniu wektora orientacji gracza:

```
1 move_direction = orientation.forward * vertical_input + orientation.right
2                                                         * horizontal_input;
```

Podstawowy kod, który przesunie gracza znajdującego na podstawowej powierzchni przedstawia się następująco:

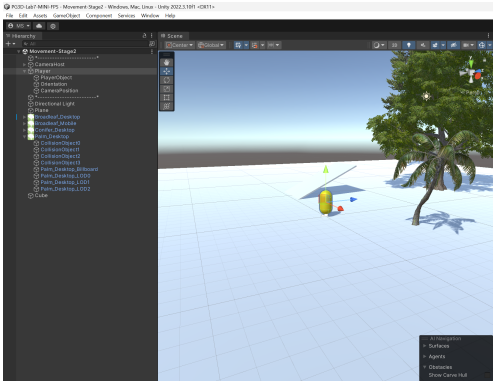
```
1 rb.AddForce(move_direction.normalized * move_speed * 10.0f, ForceMode.Force);
```

Wartość 10.0f to stała określająca podstawową prędkość gracza.

Bardzo ważnym elementem jest hierarchia obiektów jaką należy zastosować do obiektu gracza, kamery oraz obiektów wspomagających, co zostało przedstawione na Rysunku 14.

13 Laboratorium nr 11 – Obsługa sieci

Realizowane są dwa zadania: pierwsze dotyczy komunikatora tekstowego, natomiast zadanie drugie wskazuje w jaki sposób realizuje się algorytm sterowania autorytatywnego za pomocą podejcia middleware. Do obu zadań najlepiej zastosować pakiet Photo PUN/Photon Fusion 2.



Rysunek 14: Hierarchia obiektów gracza w projekcie kontrolera FPS

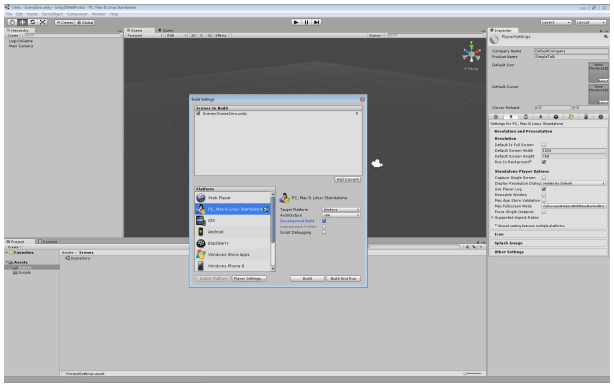
13.1 Obsługa sieci – komunikator tekstowy w Unity

Komentarz 13.1 *Podobnie jak w zadaniu w optymalizacji sceny graficznej, obsługa sieci jest również ściśle zależna od numeru wersji Unity. Dlatego, zalecane jest zastosowanie systemu/pakietu PUN 2 - FREE, który współpracuje z najnowszymi wersjami Unity.*

Głównym zadaniem tego laboratorium będzie zbudowanie nieskomplikowanego komunikatora sieciowego. Komunikaty tekstowe będą przesyłane do wszystkich osób podłączonych do głównego serwera komunikatora. Opracowana aplikacja, powinna pozwolić uruchomić się w trybie serwera bądź klienta, podać adres IP serwera, nick. W przypadku serwera oraz klienta, powinna istnieć możliwość zamknięcia połączenia tj. odłączenie się od serwera, czy ekranu klienta i powrót do okna startowego aplikacji. Do realizacji można wykorzystać starszą wersję Unity np. 5.6, lub pierwsze wydania 2017.x gdzie dostępna jest jeszcze starsza wersja obsługi sieci z komponentem NetworkView.

Niestety, nowsze rozwiązanie UNet, również jest określone jako przestarzałe, choć można ja zastosować do opracowania zadania. Jednakże, zamiast UNet lepiej jest zastosować pakiet PhotonNetwork, który w wersji darmowej oferuje odpowiednie API do obsługi sieci, i współpracuje z najnowszymi oraz starszymi wersjami Unity.

Sprawdzenie poprawności otrzymanego programu należy zrealizować korzystać ze dodatkowej zbudowanej wersji binarnej aplikacji. Proces budowy jest wykonywany za pomocą opcji Build z menu File, Rys. 15. Należy jednak wykonać dwie dodatkowe czynności. Po pierwsze, należy dodać do listy Scenes in Build za pomocą przycisku Add current, scenę jaka zawiera implementacje komunikatora. Następnie, należy zaznaczyć opcję „Run In Background” w opcjach, jakie ukrywają się pod przyciskiem Player Settings. Opcja Run In Background oznacza, że jeśli aplikacja nie będzie aktywna (np. przesłonięta przez okno innej aplikacji), to nadal będzie aktywnie przetwarzać wszystkie zdarzenia.



Rysunek 15: Parametry budowy aplikacji w postaci binarnej

13.1.1 Uwagi związane z zadaniem dla starszych wersji Unity

W najprostszej postaci komunikator wymaga obecności tylko obiektu kamery oraz obiektu np.: MainLogicObject, do którego zostanie podłączony skrypt do obsługi komunikatów oraz komponenty Network View.

Skrypt do obsługi komunikatora sieciowego może przyjąć następującą postać.

```
using UnityEngine;
using System.Collections;

public class NetworkCode : MonoBehaviour {
    public string connectionIP = "127.0.0.1";
    public int connectionPort = 25001;
    public int maxConnections = 15;

    private static ArrayList messages = new ArrayList();
    private Vector2 scrollView = Vector2.zero;

    private string message;

    void Start () {
        message = "" ;
    }

    void OnGUI() {
        if (Network.peerType == NetworkPeerType.Disconnected) {
            GUI.Label(new Rect(10, 10, 200, 20), "Status: Disconnected");
            if (GUI.Button(new Rect(10, 30, 120, 20), "Client Connect")) {
                Network.Connect(connectionIP, connectionPort);
            }
            if (GUI.Button(new Rect(10, 50, 120, 20), "Initialize Server")) {
                Network.InitializeServer(maxConnections, connectionPort, false);
            }
        }

        if (Network.peerType == NetworkPeerType.Server) {
            GUI.Label(new Rect(10, 10, 200, 20), "Run as server");
        }

        if (Network.peerType == NetworkPeerType.Client) {
            GUI.Label(new Rect(10, 10, 300, 20), "Status: Connected as Client");

            if (GUI.Button(new Rect(10, 30, 120, 20), "Disconnect")) {
                Network.Disconnect(200);
            }

            message = GUI.TextField(new Rect(0, 100, 150, 25), message);
            if (GUI.Button(new Rect(150, 200, 50, 25), "Send")) {
                networkView.RPC("ReciveMessage", RPCMode.All, message);
                message = "";
            }
        }
        GUILayout.BeginArea(new Rect(0 , 120 , 400 , 200));
        scrollView = GUILayout.BeginScrollView(scrollView);
        foreach(string c in messages) {
            GUILayout.Label(c);
        }
        GUILayout.EndArea();
        GUILayout.EndScrollView();
    }

    [RPC]
    private void ReciveMessage(string sentMess) {
        messages.Add(sentMess);
    }
}
```

13.1.2 Fragmenty kodu dla pakietu PUN 2 - Free

Poniżej podano fragmenty kodu realizującego przykład „chatu" w sposób podobny do technologii opartej o NetworkView, ale przy wykorzystaniu pakietu PhotonNetwork.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Photon.Pun;
using Photon.Realtime;

using UnityEngine.UI;

public class NetLogicScript : MonoBehaviourPunCallbacks {
        public Text chatextwidget;
        public InputField inputText;
        public InputField userName;

        //PhotonView myView;

// Start is called before the first frame update
void Start() {
        chatextwidget.text = "";
        //myView=GetComponent<PhotonView>();
}

// Update is called once per frame
void Update() {

}

public void OnConnectBTN() {
        PhotonNetwork.ConnectUsingSettings();
}

public void OnDisconnectBTN()
{
        if(PhotonNetwork.IsConnected) {
                PhotonNetwork.LeaveRoom();
                PhotonNetwork.Disconnect();
        }
        else {
                Debug.LogWarningFormat("Net1-Ex1: program is not coneected to Photon server!");
        }
}

public void OnSendBTN()
{
        if(photonView != null)
        {
                //if(photonView.IsMine)
                {
                        string messageToSend = "<b>" + userName.text +
                           "</b>" + ":" + inputText.text;
                        photonView.RPC("ReceiveText", RpcTarget.All, messageToSend);
                }
        //else {
        //    Debug.Log("Net1-Ex1: photonView is not mine");
        //}
        }
        else {
                Debug.Log("Net1-Ex1: photonView is null");
        }
}

[PunRPC]
void ReceiveText(string newMessage)
{
        string chatRoomText = chatextwidget.text + newMessage + "\n";
        chatextwidget.text = chatRoomText;
        Debug.Log("Net1-Ex1: ReceiveText is called");
}

public override void OnConnectedToMaster() {
        Debug.Log("Net1-Ex1: OnConnectedToMaster() was called by PUN");
```

```
72
73          GameObject.Find("TextInputField").GetComponent<InputField>().interactable = true;
74          GameObject.Find("SendTextBTN").GetComponent<Button>().interactable = true;
75
76          PhotonNetwork.JoinOrCreateRoom("ChatRoom", new RoomOptions { MaxPlayers = 16 }, null);
77  }
78
79  public override void OnDisconnected(DisconnectCause cause) {
80          Debug.Log("Net1-Ex1: OnDisconnected() was called by PUN");
81
82          GameObject.Find("TextInputField").GetComponent<InputField>().interactable = false;
83          GameObject.Find("SendTextBTN").GetComponent<Button>().interactable = false;
84  }
85
86
87  }
```

13.2 Obsługa sieci – prototyp gry sieciowej dla kilkunastu graczy

Zadanie realizowane na tym laboratorium można oprzeć o Lab. Nr 8, gdzie można wykorzystać zaprojektowany labirynt jako miejsce toczącej się gry. Zalecane jest, aby system przekazu komunikatów został wbudowany w prototyp rozwiania omawianego w ramach tego zadania. Projekt zrealizowany na Laboratorium ma pozwalać na poruszanie się graczy w wcześniej przygotowanym obszarze. Dodatkowo, każdy z graczy ma mieć możliwość kreacji podstawowych obiektów np.: sześcianu czy kapsuły. Nowo powstałe obiekty, co oczywiste, muszą pojawiać się u wszystkich graczy w sieci.

13.2.1 Uwagi związane z zadaniem

Analogiczne jak poprzednio należy utworzyć pusty projekt oraz przynajmniej jedną scenę. W przykładowej scenie umieszcza się tylko dwa główne obiekty, jeden o nazwie Arena, zawierający wszystkie elementy obszaru, w którym toczy się gra. Może to być obszar w postaci obiektu terrain, a także inne obiekty 3D, czy też elementy sceny np.: światła. Drugim istotnym obiektem będzie SpawnPoint. Gracze, którzy podłączą się do gry będą ją rozpoczynać w punkcie o tej nazwie. Do tego obiektu podobnie jak poprzednio podłączono skrypt z klasą NetworkLogic. Podłączona zostanie także kamera, choć nie jest ona potrzebna. Po zalogowaniu się do gry, kamera podłączona do punktu SpawnPoint zostanie wyłączona, a włączymy kamerę gracza.

Istotna różnica polega na tym, iż nie jest tworzony obiekt gracza, ponieważ będzie on dynamicznie kreowany po zalogowaniu się graczy na serwer. Należy jednak utworzyć taki obiekt, w postaci sześcianu, bądź też kapsuły. Można też wykorzystać gotowe rozwiązania z pakietu Character Controller. Wymaga ono jednak pewnej zmiany dotyczącej sposobu sterowania.

Skrypt odpowiadający za obsługę podstawowej logiki podłączony do SpawnPoint:

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class NetworkLogic : MonoBehaviour {
5
6           public GameObject PlayerPrefab;
7
8           public string connectionIP = "127.0.0.1";
9           public int connectionPort = 25001;
10          public int maxConnections = 15;
11
12          public bool playerConnected;
13          private int groupID = 1;
14
15
16          void Start () {
17                  playerConnected = false;
18          }
19
20          void CreateNetworkPlayer() {
21                  playerConnected = true;
22                  var g = (GameObject)Network.Instantiate(PlayerPrefab,
23                          transform.position, transform.rotation, groupID);
24                  var obj = g.GetComponentInChildren<Camera>();
```

```
                obj.camera.enabled = true;
                camera.enabled = false;
        }

        void OnDisconnectedFromServer() {
                playerConnected = false;
        }
        void OnPlayerDisconnected(NetworkPlayer player) {
                Network.DestroyPlayerObjects( player );
        }
        void OnConnectedToServer() {
                CreateNetworkPlayer();
        }
        void OnServerInitialized() {
                CreateNetworkPlayer();
        }
        void OnGUI () {
                // menu wyboru, czy program
                // działa w trybie serwera, czy klienta
        }
}
```

Niezbędne jest też wniesienie poprawki do skryptu kontrole obsługującego wejście. Znajdują się w pakiecie Character Controller[14]. W metodzie Update należy na początku dopisać dwie linie kodu:

```
if( !networkView.isMine )
    return;
```

Powyższa konstrukcja zapewnia, że informacje o wciśniętych klawiszach, które są przetwarzane przez pozostałe skrypty obiektu gracza będą przesyłane tylko do właściciela obiektu networkView. Przekłada się to na fakt, że lokalny klient (a także serwer), nie będzie przekazywał informacji np.: o spacji, która oznacza skok gracza, do pozostałych obiektów graczy.

Bezpośrednie podejście zrealizowane przy pomocy NetworkView, może okazać się dalece niewystarczające ze względu na braki w synchronizacji pozycji graczy. Jednym z możliwych rozwiązań jest zastosowanie interpolacji. Wymaga to zmiany w tworzeniu gracza, bowiem należy dodać dodatkowy kod różny dla serwera oraz klienta:

```
        void CreateNetworkPlayer(int playerType)
        {
                playerConnected = true;
                var g = (GameObject)Network.Instantiate(PlayerPrefab,
                        transform.position, transform.rotation, groupID);


                if(playerType == 0)
                        g.AddComponent("NetworkPlayerLogicServer");

                if(playerType == 1)
                        g.AddComponent("NetworkPlayerLogicRemote");


                var obj = g.GetComponentInChildren<Camera>();
                obj.camera.enabled = true;
                camera.enabled = false;
        }
```

W przypadku serwera wymagana jest serializacja danych realizowana w następujący sposób:

```
using UnityEngine;
using System.Collections;

public class NetworkPlayerLogicServer : MonoBehaviour {


        void OnSerializeNetworkView( BitStream stream, NetworkMessageInfo info )
        {
                Vector3 position = Vector3.zero;
                Quaternion rotation = Quaternion.identity;

```

[14] Uwaga,skrypt kontrolera, może mieć nieco inną postać niż kontroler omawiamy w tym zadaniu.

```
12                  if( stream.isWriting )
13                  {
14                          position = transform.position;
15                          rotation = transform.rotation;
16
17                          stream.Serialize( ref position );
18                          stream.Serialize( ref rotation );
19                  }
20                  else
21                  {
22                          stream.Serialize( ref position );
23                          stream.Serialize( ref rotation );
24
25                          transform.position = position;
26                          transform.rotation = rotation;
27                  }
28
29          }
30  }
```

Dla serwera nie ma żadnych opóźnień, więc proces synchronizacji nie jest potrzebny. Inaczej należy postąpić w przypadku klienta, gdzie dodatkowy kod wykonuje interpolacje pozycji gracza (uwaga: w przypadku stosowania kontrolera gracza z Character Controller może to nie wystarczyć, aby uzyskać płynną animacje postaci graczy).

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class NetworkPlayerLogicRemote : MonoBehaviour {
5   public float InterpolationBackTime = 0.1f;
6
7   private playerState[] stateBuffer = null;
8   private int stateCount;
9
10  private struct playerState
11  {
12          public Vector3 Position;
13          public Quaternion Rotation;
14          public double Timestamp;
15
16          public playerState( Vector3 pos, Quaternion rot, double time )
17          {
18                  this.Position = pos;
19                  this.Rotation = rot;
20                  this.Timestamp = time;
21          }
22  }
23
24
25  void Start()
26  {
27          stateBuffer = new playerState[ 25 ];
28          stateCount = 0;
29  }
30
31
32  void Update()
33  {
34          if( networkView.isMine ) return;
35          if( stateCount == 0 ) return;
36
37
38          double currentTime = Network.time;
39          double interpolationTime = currentTime - InterpolationBackTime;
40
41          if( stateBuffer[ 0 ].Timestamp > interpolationTime )
42          {
43                  for( int i = 0; i < stateCount; i++ )
44                  {
45                          if( stateBuffer[ i ].Timestamp <= interpolationTime
46                                  || i == stateCount - 1 )
47                          {
48
```

```
49                                  playerState startState = stateBuffer[ i ];
50                                  playerState targetState =
51                                          stateBuffer[ Mathf.Max( i - 1, 0 ) ];
52
53
54                                  double length = targetState.Timestamp - startState.Timestamp;
55                                  float t = 0f;
56                                  if( length > 0.0001 )
57                                  {
58                                    t = (float)( ( interpolationTime - startState.Timestamp )
59                                          / length );
60                                  }
61
62                                  transform.position =
63                                  Vector3.Lerp( startState.Position, targetState.Position, t );
64                                  transform.rotation = Quaternion.Slerp(startState.Rotation,
65                                          targetState.Rotation, t);
66
67                                  return;
68                          }
69                  }
70          }
71          else
72          {
73                  double extrapolationLength = (interpolationTime - stateBuffer[0].Timestamp);
74                  if (extrapolationLength < 1 && stateCount > 1 )
75                  {
76                                  transform.position = stateBuffer[0].Position +
77                                          (((stateBuffer[0].Position - stateBuffer[1].Position) /
78                                                  ((float)stateBuffer[0].Timestamp -
79                                                          (float)stateBuffer[1].Timestamp) )
80                                                                  * (float)extrapolationLength);
81                                  transform.rotation = stateBuffer[0].Rotation;
82                  }
83          }
84  }
85
86  void OnSerializeNetworkView( BitStream stream, NetworkMessageInfo info )
87  {
88          if( stream.isWriting )
89          {
90                  Vector3 position = transform.position;
91                  Quaternion rotation = transform.rotation;
92
93                  stream.Serialize( ref position );
94                  stream.Serialize( ref rotation );
95          }
96          else
97          {
98                  Vector3 position = Vector3.zero;
99                  Quaternion rotation = Quaternion.identity;
100
101                 stream.Serialize( ref position );
102                 stream.Serialize( ref rotation );
103
104
105                 for (int k=stateBuffer.Length-1;k>0;k--)
106                 {
107                         stateBuffer[k] = stateBuffer[k-1];
108                 }
109
110                 stateBuffer[0] = new playerState( position, rotation, info.timestamp) ;;
111
112                 stateCount = Mathf.Min(stateCount + 1, stateBuffer.Length);
113         }
114 }
115 }
```

13.3 Obsługa sieci – prototyp gry sieciowej – sterowanie autorytatywne

Komentarz 13.2 *Podobnie jak w zadaniu w optymalizacji sceny graficznej, obsługa sieci jest również ściśle zależna od numeru wersji Unity. Dlatego, zalecane jest zastosowanie systemy/pakietu PUN 2 - FREE, który współpracuje z najnowszymi wersjami Unity. Numer wersji jest określany na początku zajęć laboratoryjnych, na początku każdego semestru.*

Zadanie realizowane na tym laboratorium stanowi bezpośrednią kontynuację zadania z poprzedniego podpunktu. Celem, jest wbudowanie tzw. sterowania autorytatywnego. Polega ono na tym, iż informacje o sterowaniu np.: jakie klawisze zostały wciśnięte są przekazywane do serwera, który następnie przekazuje informacje do wszystkich klientów którzy lokalnie przeprowadzają proces symulacji zachowania się gracza. Dodatkowo, przekazywana jest też informacja o położeniu obiektów, i w przypadku dużych opóźnień w komunikacji, przeprowadzana jest interpolacja graczy zdalnych.

13.3.1 Uwagi związane z zadaniem

Analogiczne jak poprzednich zadaniach należy utworzyć pusty projekt oraz przynajmniej jedną scenę. W przykładowej scenie umieszcza się tylko dwa główne obiekty, jeden o nazwie Arena, zawierający wszystkie elementy obszaru, w którym toczy się gra. Drugim istotnym obiektem pozostaje SpawnPoint. Gracze, którzy podłączą się do gry będą ją rozpoczynali w punkcie o tej nazwie. Do tego obiektu podobnie jak poprzednio podłączono skrypt z klasą NetworkLogic. Podłączona zostanie także kamera, choć nie jest ona potrzebna. Po zalogowaniu się do gry, kamera podłączona do punktu SpawnPoint zostanie wyłączona, a włączymy kamerę gracza.

Istotna różnica polega na tym, iż nie jest tworzony obiekt gracza, ponieważ będzie on dynamicznie kreowany po bądź zalogowaniu się graczy na serwer. Należy jednak utworzyć taki obiekt, w postaci sześcianu, bądź też kapsuły. Można też wykorzystać gotowe rozwiązania z pakietu Character Controller. Wymaga ono jednak znacznie większych zmian dotyczącej sposobu sterowania. Niezwykle ważne zmiany dotyczą obiektu Network View, w którym należy wyłączyć przesyłanie informacji o stanie obiektów.

Zmiany w oryginalnym skrypcie FPSInputController.js (lub nowszym napisanym w C#) polegają na sprawdzeniu czy jesteśmy właścicielem widoku sieciowego, i w zależności od tego faktu odebranie bądź nie informacji o sterowaniu z systemu wejścia Input:

```
1  @System.NonSerialized
2  var motor : CharacterMotor;
3
4  @System.NonSerialized
5  var directionVector : Vector3;
6
7  @System.NonSerialized
8  var inputJump : boolean;
9
10 function Awake () {
11         motor = GetComponent(CharacterMotor);
12 }
13
14 // Update is called once per frame
15 function Update () {
16         if( networkView.isMine )
17         {
18                 directionVector = new Vector3(Input.GetAxis("Horizontal"),
19                         0, Input.GetAxis("Vertical"));
20                 inputJump = Input.GetButton("Jump");
21         }
22 }
23
24 function Simulate() {
25
26         if (directionVector != Vector3.zero) {
27
28                 var directionLength = directionVector.magnitude;
29
30                 directionVector = directionVector / directionLength;
31
32                 directionLength = Mathf.Min(1, directionLength);
33
34                 directionLength = directionLength * directionLength;
35
36                 directionVector = directionVector * directionLength;
37         }
```

```
38
39          motor.inputMoveDirection = transform.rotation * directionVector;
40          motor.inputJump = inputJump;
41
42  }
43
44  @script RequireComponent (CharacterMotor)
45  @script AddComponentMenu ("Character/FPS Input Controller")
```

Najważniejsze zmiany w skrypcie NetworkedPlayer również są dość obszerne. Należy dodać dwa obiekty networkState (stosowany do interpolacji) oraz np.: remoteCommand, gdzie przesyłane będą informacje odnoszące się do sterowania gracza zdalnego. Główny przetwarzane informacji odbywa się w metodzie FixedUpdate, gdzie odbierane są komunikaty, tworzona jest historia poleceń zdalnych oraz zlecane jest wywołanie funkcji sterującej ProcessInput dla poszczególnych klientów podłączonych dla serwera.

```
1   void FixedUpdate()
2   {
3           if( networkView.isMine )
4           {
5                   remoteMove remote_move = new remoteMove( playerRemote.directionVector,
6                           playerRemote.inputJump, Network.time );
7
8                   remoteMoveHistory.Insert( 0, remote_move );
9
10                  if( remoteMoveHistory.Count > 200 )
11                  {
12                          remoteMoveHistory.RemoveAt( remoteMoveHistory.Count - 1 );
13                  }
14
15                  playerRemote.Simulate();
16
17                  networkView.RPC( "ProcessInput", RPCMode.Server,
18                          remote_move.directionVector, remote_move.inputJump,
19                                  transform.position );
20          }
21  }
22
23  [RPC]
24  void ProcessInput( Vector3 dirVector, bool inJump, Vector3 position,
25                          NetworkMessageInfo info ) {
26          if( networkView.isMine ) return;
27
28          if( !Network.isServer ) return;
29
30          playerRemote.directionVector = dirVector;
31          playerRemote.inputJump = inJump;
32          playerRemote.SimulateInput();
33
34          if( Vector3.Distance( transform.position, position ) > 0.1f ) {
35                  networkView.RPC( "CorrectPlayerState", info.sender, transform.position );
36          }
37  }
```

14 Laboratorium nr 12 – Obsługa shaderów

Zadanie realizowane na laboratorium dotyczy utworzenia kilku typów shaderów, ponieważ środowisko Unity wykorzystuje kilka różnych podsystemów OpenGL, Metal bądź DX11/DX12, to głównym językiem opisu shaderów będzie język CG (przykładowy shader można odszukać na stronach dokumentacji np.: https://docs.unity3d.com/Manual/built-in-shader-examples-vertex-data.html). Realizując zadanie można również skorzystać z rozwiązania o nazwie Unity Shader Graph.

Dokonać implementacji następujących shaderów bezpośrednio za pomocą języka CG:

(a) shader tworzący jednolity kolor,

(b) shader tworzący planszę szachową,

(c) zaimplementować shadery podane jako przykład w dokumentacji, do wizualizacji współrzędnych UV, kolorowania wierzchołków, wizualizacji normalnych, oraz stycznych i binormalnych,

(d) zbiór Mandelbrota oraz zbiór Juli,

(e) wykonać port shadera Heart do środowiska Unity ze strony: https://www.shadertoy.com/view/XsfGRn.

14.1 Uwagi związane z zadaniem

Podstawowe wersje wymienionych shaderów można odszukać w materiałach z wykładu Nr 7 pt. „Shadery oraz rozszerzanie Unity".

15 Laboratorium nr 13 – Obiekty tworzone proceduralnie

Główne zadanie tego laboratorium to zbudowanie w sposób proceduralny – w trakcie tzw. czasu run-time – następujących obiektów siatkowych:

(a) płaska siatka (grid), inaczej mówiąc obiekt płaszczyzny o wskazanych wymiarach,

(b) sześcian,

(c) czworościan,

(d) sfera,

(e) zadanie dodatkowe/opcjonalne, jednakże bardzo zalecane do wykonania, to wykorzystanie krzywych parametrycznych (ang. spline), do budowy ścieżki lub drogi np. w oparciu o komponent terenu.

Utworzone obiekty oprócz podstawowej siatki geometrycznej, muszą też posiadać współrzędne teksturowanie oraz poprawnie skierowane wektory normalne oraz właściwy opis stycznych.

15.1 Uwagi związane z zadaniem

Informacje o tworzeniu obiektu siatki i czworościanu zostały podane na wykładzie Nr 9, pt. Anatomia klasy/obiektu MESH.

16 Laboratorium nr 14 – Krzywe sklejane – do opracowania

16.1 Uwagi związane z zadaniem

17 Inne informacje

Przegląd pozycji książkowych dotyczących środowiska Unity w języku angielskim można odszukać na stronach wydawnictw Apress, czy PackPub. Poniższy spis literatury choć odnosi się do wszystkich ćwiczeń oraz zadań z przedmiotu Programowanie gier 3D, i jest tylko arbitralnie dokonanym wyborem pozycji związanych ze środowiskiem Unity. Ze względu na dynamiczny rozwój środowiska Unity, najnowsze API lub rewizje poszczególnych podsystemów są prezentowane za pośrednictwem dokumentacji na stronach producenta środowiska Unity.

Literatura

[1] Materiały dostępne na stronie głównej Unity, ostatni dostęp 05.11.2023, https://docs.unity.com/.

[2] John Bach, Unity 2d game development: Beginner's Guide to 2D game development with Unity, Mem Lnc, 2020, https://www.amazon.com/Unity-2d-game-development-Beginners/dp/B08BW5Y6MY.

[3] Sue Blackman and Adam Tuliper, Learn Unity for Windows 10 Game Development, Apress, 2016, https://www.apress.com/gp/book/9781430267584.

[4] Jeremy G. Bond, Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#, 2nd Edition, Addison-Wesley Professional, 2017, https://www.pearson.com/us/higher-education/program/Gibson-Bond-Introduction-to-Game-Design-Prototyping-and-Development-From-Concept-to-Playable-Game-with-Unity-and-C-2nd-Edition/PGM1237076.html.

[5] Nicolas A. Borromeo, Hands-On Unity 2020 Game Development, Packt Publishing, 2020, https://www.packtpub.com/product/hands-on-unity-2020-game-development/9781838642006

[6] John P. Doran, Building an FPS Game, Packt Publishing, 2015, https://www.oreilly.com/library/view/building-an-fps/9781782174806/

[7] Harrison Ferrone, Learning C# by Developing Games with Unity 2020 - Fifth Edition, Packt Publishing, 2020, https://www.packtpub.com/product/learning-c-by-developing-games-with-unity-2020-fifth-edition/9781800207806

[8] Mike Geig, Unity. Przewodnik projektanta gier. Wydanie III, Helion, 2019, https://helion.pl/ksiazki/unity-przewodnik-projektanta-gier-wydanie-iii-mike-geig,unipp3.htm#format/d

[9] Will Goldstone, Projektowanie gier w środowisku Unity 3.x, Helion, 2012, http://helion.pl/ksiazki/projektowanie-gier-w-srodowisku-unity-3-x-will-goldstone,prgun3.htm

[10] Casey Hardman, Game Programming with Unity and C#, Apress, 2020, https://www.apress.com/gp/book/9781484256558

[11] Alex Okita, Learning C# Programming with Unity 3D, second edition, CRC Press, 2019, https://www.routledge.com/Learning-C-Programming-with-Unity-3D-second-edition/Okita/p/book/9781138336810

[12] Jacek Ross, Unity i C#. Praktyka programowania gier, Helion, 2020, https://helion.pl/ksiazki/unity-i-c-praktyka-programowania-gier-jacek-ross,uncppg.htm#format/d

[13] Francesco Sapio, Francesco SapioUnity. Przepisy na interfejs gry, Helion, 2016, https://helion.pl/ksiazki/unity-przepisy-na-interfejs-gry-francesco-sapio,unipig.htm#format/d

[14] Matt Smith, Unity 2018 Cookbook - Third Edition, Packt Publishing, 2018, https://subscription.packtpub.com/book/game-development/9781788471909

[15] Alan R. Stagner, Unity Multiplayer Games, Pack Pub, 2013, http://www.packtpub.com/unity-multiplayer-games/book

[16] Kelvin Sung and Greg Smith, Basic Math for Game Development with Unity 3D, Apress, 2019, https://www.apress.com/gp/book/9781484254424

[17] Alan Thorn, Unity i Blender. Praktyczne tworzenie gier, Helion, 2015, https://helion.pl/ksiazki/unity-i-blender-praktyczne-tworzenie-gier-alan-thorn,unible.htm#format/e