Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Programowanie gier 3D – Laboratorium

Programowanie gier 3D – lista zadań laboratoryjnych – w trakcie tworzenia!

1 Wprowadzenie

Środowisko Unity3D (http://unity3d.com), to środowisko pracy do rozwoju interaktywnych aplikacji z grafiką 3D oraz 2D. Istotną cechą środowiska Unity3D jest multi-platformowość. Oprócz dwóch wersji środowiska Unity3D dla systemów Windows oraz Mac OS, Unity3D pozwala na utworzenie postaci finalnej projektowanej aplikacji dla kilku różnych platform sprzętowo-programowych min. Windows, Linux, Android, IOS.

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

(a) Shiva3D, http://stonetrip.com/,

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

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

(d) Torque3D, http://www.garagegames.com/,

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

(f) UDK, http://www.unrealengine.com/udk/.

W przypadku Unity3D oprócz informacji oraz dokumentacji na strona producenta:

(a) podstawowe informacje o użytkowaniu środowiska Unity3D – http://docs.unity3d.com/Documentation/Manual/index.html,

(b) opis komponentów z których tworzone są aplikację w Unity3D – http://docs.unity3d.com/Documentation/Components/index.html,

(c) informacje o API i programowaniu w Unity3D – http://docs.unity3d.com/Documentation/ScriptReference/index.html.

Należy też wskazać portal Unity3D Community – jest to strona społeczności min. z dodatkowymi skryptami: http://wiki.unity3d.com/index.php/Main_Page. Przydatne informacje znajdują się także na stronie: http://www.unity3dstudent.com.

2 Spis zadań do zrealizowana

2.1 Laboratorium nr 1 – Podstawy środowiska Unity3D

Głównym celem tego zadania[1] jest zapoznanie się z podstawowymi elementami środowiska Unity3D:

- okno główne aplikacji, konfiguracja okien roboczych Unity3D, zasoby „Asset Store”,
- rola okna Inspector, Asset, Hierarchy,
- główne komponenty dostępny w ramach systemu Unity3D (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: http://unity3d.com/learn/tutorials/modules/beginner/editor.

[1] To zadanie oraz przyszłe zostały opracowane na podstawie pozycji wymienionych w literaturze.

2.2 Laboratorium nr 2 – „Tocząca się kulka”

Zaprojektować na podstawie materiału „Roll a ball”:

- http://unity3d.com/learn/tutorials/projects/roll-a-ball),

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.

2.2.1 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 świtała,
- obiekt Input (metody: GetAxis, GetKeyUp),
- obiekt transform (metoda Rotate, własność position), rigidbody (metoda AddForce),
- śledzenie kamerą obiektu gracza.

2.3 Laboratorium nr 3 – Edycja terenu

Zadania jakie należy wykonać:

- utworzyć projekt zawierający teren (lub kilka komponentów typu teren)
- teren ma zawierać różne teksty reprezentujące, skały, trawę, piasek,
- dodać pojedyncze drzewa a także grupy drzew oraz trawę przy wykorzystaniu mechanizów Unity3D,
- opcjonalnie wykorzystując „Tree Creator”, dołączyć drzewa reagujące na wiatr,
- dodać tzw. skybox reprezentujący niebo nad terenem,
- wykorzystując gotowy komponent Unity3D o nazwie FPS Character (zawarty w Standard Asset), umożliwić graczowi poruszanie się po terenie,
- dodać element GUI pokazujący liczbę klatek na sekundę,
- zbudować końcową postać aplikacji.

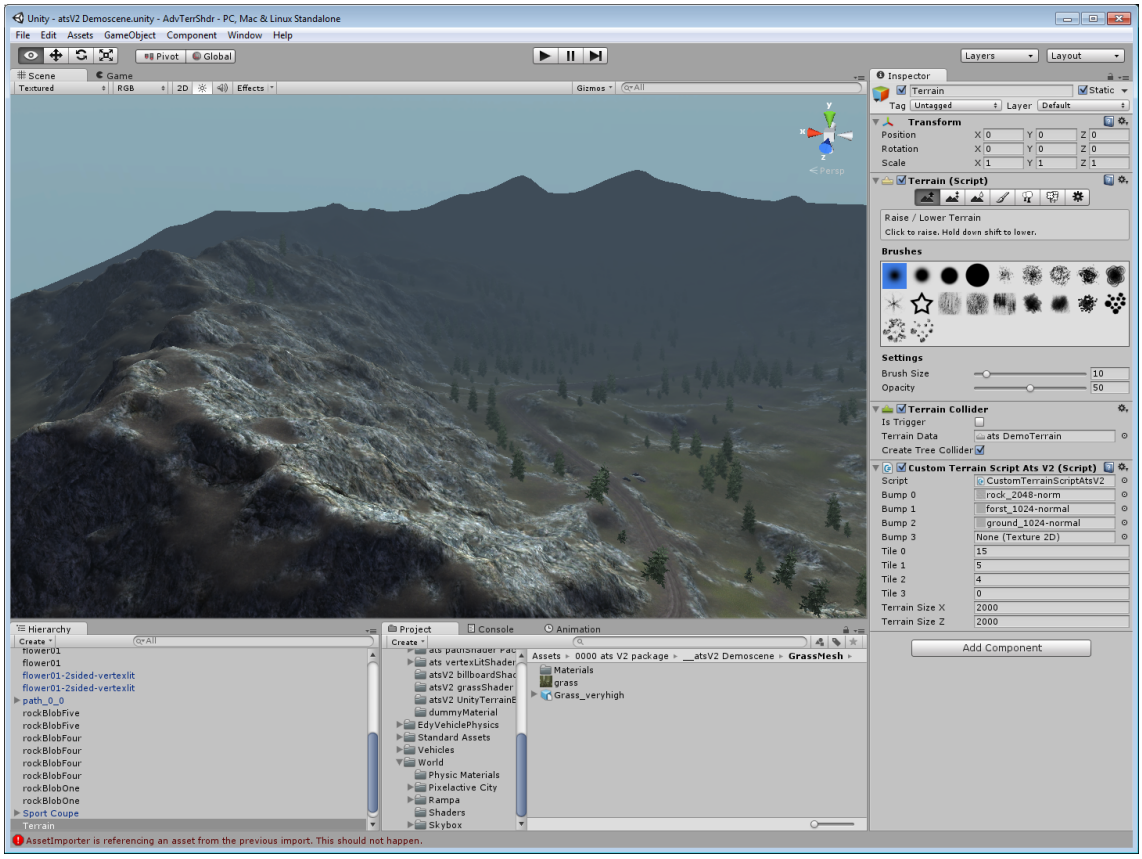
2.3.1 Uwagi związane z edycją terenu

Ćwiczenie odnoszące się do komponenty teren (ang. terrain) w środowisku Unity3D. Czynności związane ze stworzeniem terenu:

- utworzyć nowy obiekt reprezentujący teren (Terrain → Create Terrain),
- ustawić wymiary X i Y terenu na 1000, natomiast wysokość na 200 (Terrain → Set Resolution),
- zastosować opcję „Flatten Heightmap” i ustawić parametr wysokość na wartość 50,
- 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 pędz a także można ustawić rozmiar pędzla jak również kształt. Uwaga, aby obniżyć wartość wysokości terenu należy „malować” mając wciśnięty klawisz shift,



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

- 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 Asset" oraz „Advanced Terrain Shader V2". Tekstura wody dostępna jest w pakiecie „Water (Basic)" dla wersji darmowej Unity3D.

Aby rozpocząć proces nakładania tekstur należy w oknie inspektora wybrać opcję malowania tekstur (ikona „Pędzla"). Klikając w przycisk „Edit Textures", można dodać zestaw tekstur do nakładania na teren. Teksturę dodajemy przyciskiem Add (podstawowy zestaw tekstur z Terrain Asset to „Good Dirt", „Grass & Rock" oraz „Grass (Hill)".

Aby dodać teksturę wody należy z okna „Project" wybrać „Water (Basic) → Daylight simple water" i umieścić na scenie. W oknie Inspector należy także ustawić odpowiednie wartości dla skali i położenia.

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.

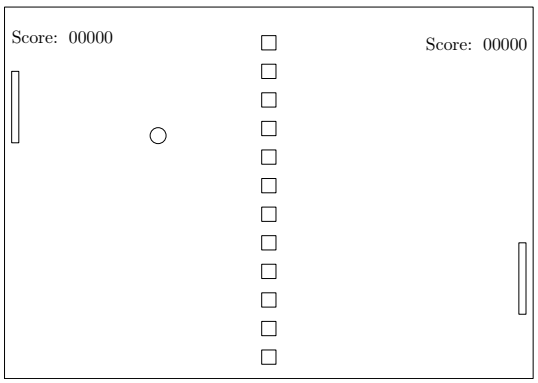
Elementem poprawiającym jakość grafiki będzie tzw. efekt flary. Należy zaimportować pakiet Light Flares. Następnie dodać do sceny światło kierunkowe (GameObject → Create Other → Directional Light). W oknie Insepctor'a niezbędna jest następująca zmiana Flare → Sun. Odpowiednio należy również zmienić pozycję oraz obrót źródła światła, aby 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 „Character Controller". Należy jednak zwrócić uwagę na kamerę, należy upewnić iż jest obecna tylko jedna kamera, należy też sprawdzić parametry obiektu repre-

zentującego gracza, aby poprawnie dobrać wartości skali, rozmiaru, czy też sposobu poruszania się.

2.4 Laboratorium nr 4 – „Realizacja gry typu Pong”

Głównym celem tego zadania jest zrealizowanie za pomocą środowiska Unity3D, 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. 2.



Rysunek 2: Ogólny koncept gry Pong

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 ma zostać zaimplementowane menu, przerwanie, opuszczenie aplikacji. Podczas zamykania aplikacji ma zostać wyświetlony dodatkowy ekran np.: informacją o autorze.

2.4.1 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:

- samodzielnie implementując ruch kulki oraz wykrywając zderzenie, obliczać wartość wektora odbicia za pomocą metody Reflect z obiektu Vector3,
- wykorzystać system symulacji fizyki dostępny w Unity3D.

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

- sterowanie zachowanie 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:

```
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          {
12                  rigidbody.AddForce( 5.0f, 5.0f, 0.0f);
13          }
14
15          void Update ()
16          {
17                  var fromVel = rigidbody.velocity;
18                  var toVel = fromVel.normalized * fltSpeed;
19                  rigidbody.velocity = Vector3.Lerp(fromVel, toVel, Time.deltaTime * fltTimeFactor);
20          }
21
22          void OnCollisionEnter(Collision col)
23          {
24                  if(col.gameObject.name == "LeftPad")
25                  {
26                          if ( col.gameObject.GetComponent<Pad>().dir_up == true)
27                          {
28                                  rigidbody.AddForce(0, 100,0);
29                          }
30                          if ( col.gameObject.GetComponent<Pad>().dir_down == true)
31                          {
32                                  rigidbody.AddForce(0, 100,0);
33                          }
34
35                  }
36          }
37
38  }
```

Skrypt sterujący paletką:

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class Pad : MonoBehaviour {
5
6           public KeyCode key_up;
7           public KeyCode key_down;
8
9           public bool dir_up;
10          public bool dir_down;
11
12          void Start () {
13                  dir_up = false;
14                  dir_down = false;
15          }
16
17          void Update () {
18
19          }
20
21          void FixedUpdate() {
22
23                  if (Input.GetKey(key_up))
24                  {
25                          transform.Translate(new Vector3(0,5.0f,0) * Time.deltaTime);
26
27                          dir_up = true;
28                  }
29                  else
30                  {
31                          dir_up = false;
32                  }
33
34                  if (Input.GetKey(key_down))
35                  {
36                          transform.Translate(new Vector3(0, 5.0f,0) * Time.deltaTime);
37
38                          dir_down = true;
```

```
39                      }
40                      else
41                      {
42                              dir_down = false;
43                      }
44              }
45
46      }
```

Implementacja sterowania paletką przez komputer można zrealizować odczytując współrzędne piłki i przesuwając paletkę w kierunku odczytanych współrzędnych:

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class AIScript : MonoBehaviour {
5
6           public float moveSpeed = 5.0f;
7
8           private GameObject ball = null;
9
10          void Start () {
11
12                  ball = GameObject.Find("Ball");
13
14          }
15
16          void LateUpdate ()
17          {
18                  if(ball.transform.position.y > transform.position.y)
19                  {
20                          transform.Translate(new Vector3(0,moveSpeed,0) * Time.deltaTime);
21                  }
22                  if(ball.transform.position.y < transform.position.y)
23                  {
24                          transform.Translate(new Vector3(0, moveSpeed,0) * Time.deltaTime);
25                  }
26          }
27  }
```

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

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

2.5 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 Unity3D (przydatny będzie darmowy pakiet z Asset Store – https://www.assetstore.unity3d.com/#/content/5154), bądź też w darmowym pakiecie NGUI.

Pakiet NGUI 2.7 można uzyskać pod adresem:

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

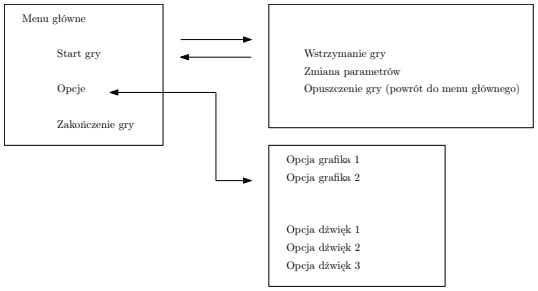
Dodatkowe informacje o pakiecie NGUI znajdują się pod adresem: http://www.tasharen.com/forum/index.php?topic=6754.

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 na opuszczenie gry. Należy przygotować menu dwujęzyczne w języku polskim oraz angielskim (z możliwością dodania później innych języków). Zakładamy iż menu wywoływane klawiszem ESC będzie znajdowała się w okno które można przesuwać np.: myszką.

Ogólny, i bardzo poglądowy schemat menu przedstawia Rys. 3.

2.5.1 Uwagi związane z zadaniem

Przydatne informacje o realizacji menu można odszukać dla klasy GUI, w dokumentacji Unity3D. Ogolnie główne komponenty jakie mogą okazać się przydatne w realizacji zadania to:



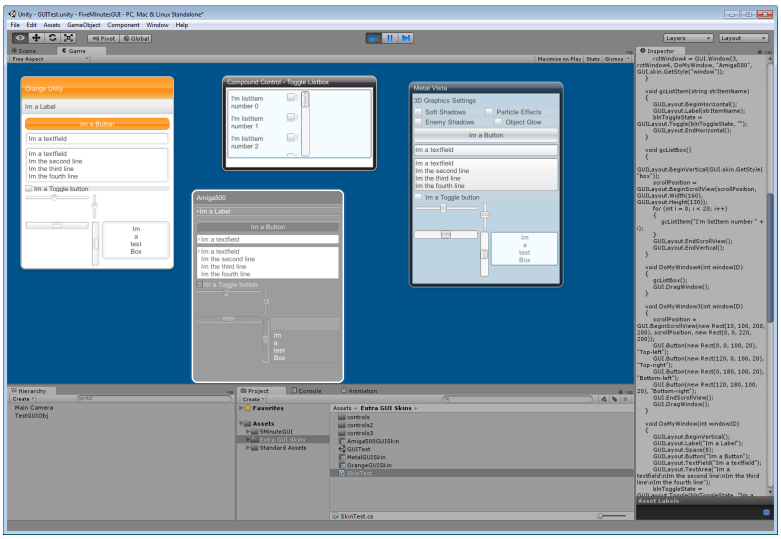
Rysunek 3: Ogólny schemat projektowanego menu w ramach zadania

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

Główną metodą w klasach obsługujących GameObject jest OnGUI, w tej metodzie powinna być realizowane obsługa interfejsu użytkownika. Klasa GUI zawiera elementy, które nie są automatycznie rozmieszczane, zawsze należy podać współrzędne podktórymi dany komponent ma się znaleźć. Klasa„ 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 Unity3D posiada także wsparcie dla stylów, darmowe style jakie można zastosować to „Extra GUI Skins" lub „Built-in GUI Skin" obydwa dostępne w Asset Store.

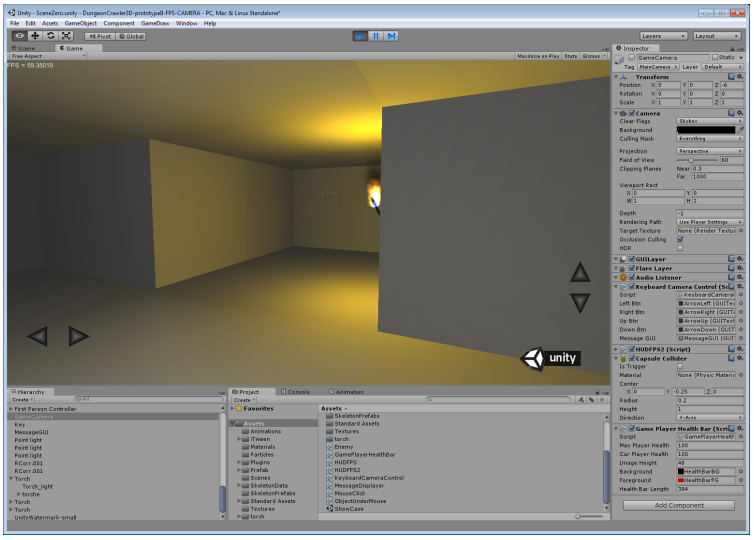


Rysunek 4: Okna uzyskane za pomocą styłów dostępnych w pakiecie „Extra GUI Skins"

2.6 Laboratorium nr 6 – „Kamera typu FPS w środowisku 3D”

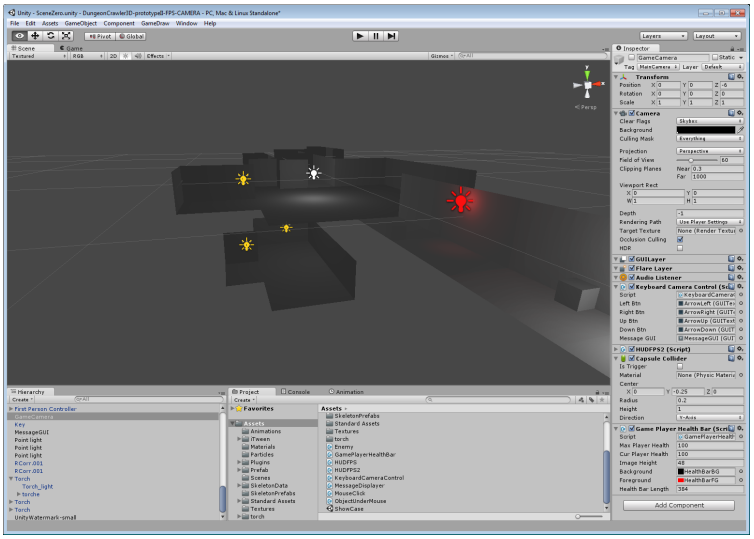
Zadanie do zrealizowania w ramach laboratorium, polega na stworzeniu prototypu gry 3D z typu FPS (ang. first-person camera). Gracz ma poruszać się swobodnie w środowisku 3D, w tym celu można wykorzystać pakiet „Character Controller”.

Poziom gry może zostać zaprojektowany z elementów przygotowanych np.: w programie Blender. Należy zwrócić uwagę na zwrot wektorów normalnych. Należy też 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://www.assetstore.unity3d.com/#/content/7275.



Rysunek 5: Projektowany poziom w grze, obiekt pochodni dostępny w Asset Store

Prototyp gry ma zwierać poziom którego konstrukcja jest zróżnicowana w pionie, może to być budynek z windą.

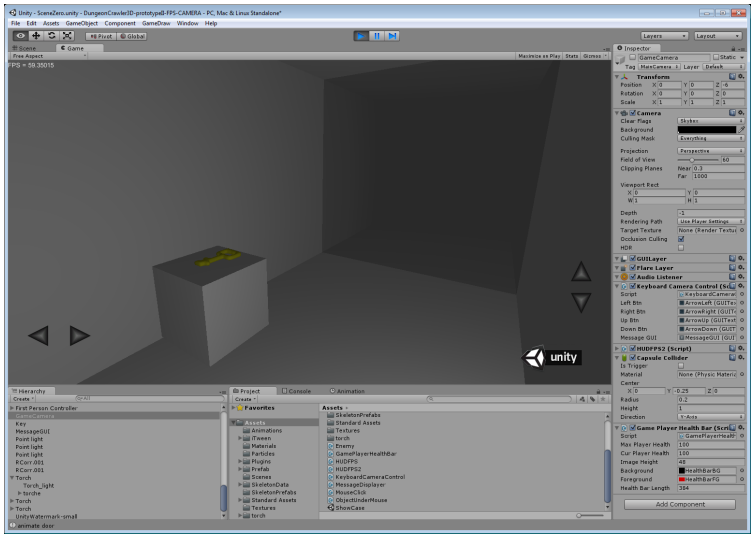


Rysunek 6: Ogólne spojrzenie na projektowany poziom w grze, wektory normalne powinny być skierowane do wewnątrz

2.6.1 Uwagi związane z zadaniem

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

```
1  if ( Input.GetMouseButtonDown(0) )
2  {
3          ray = Camera.main.ScreenPointToRay (Input.mousePosition);
4
5          if (Physics.Raycast (ray, out hit, 100.0f))
6          {
7                  if( hit.collider.name == "Door")
8                  {
9                          print("animate door");
10                         Door.animation.Play("OpenDoor");
11                 }
12                 // testy dla innych obiektów
13         }
14
15 }
```



Rysunek 7: W prototypie gry powinny znaleźć się także drzwi oraz obiekty które można zbierać np.: klucz

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 w wywołana gdy mysz znajdzie się nad obiektem oraz OnMouseLeave wywołana w momencie gdy kursor myszu opuści obszar obiektu:

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class ObjectUnderMouse : MonoBehaviour {
5
6          void OnMouseEnter()
7          {
8          renderer.material.color = new Color(1.0f, 0.0f, 0.0f);
9          }
10
11         void OnMouseExit()
12         {
13         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;
        print("animate switch 1");
        ElevatorSwitch.animation.Play("ActiveSwitch");

        ElevatorIsMoving=true;
        Invoke("ElevatorMoveDown", 1.1f);
}

if( hit.collider.name == "ElevatorSwitch" && ElevatorSwitch.animation.isPlaying == false
        && ElevatorSwitchActive == true)
{
        ElevatorSwitchActive=false;
        print("animate switch 2");
        ElevatorSwitch.animation.Play("ActiveSwitch2");

        ElevatorIsMoving=true;
        Invoke("ElevatorMoveUp", 1.1f);
}
```

Zależy to także od aktualnej pozycji windy, czy znajduje się na „piętrze" czy też na „parterze". Przesunięcie windy jest realizowana za pomocą pakietu iTween https://www.assetstore.unity3d.com/#/content/84, i po zakończeniu procesu przesunięcia odpowiednich obiektów, wywołana zostanie metoda MoveComplete.

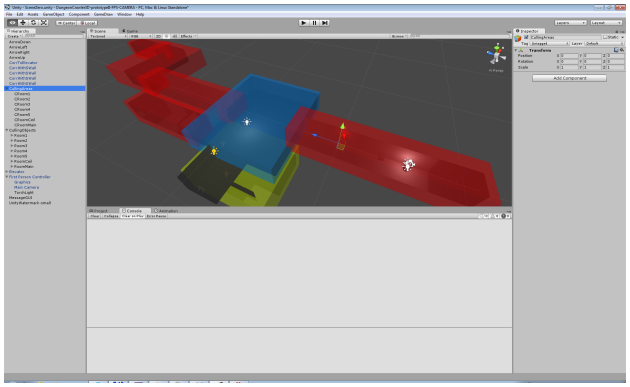
```
private void ElevatorMoveUp()
{
        iTween.MoveBy (Elevator, iTween.Hash("z", 4.0f, "time", 8, "delay", 0.0, "looptype",
                iTween.LoopType.none) );
        iTween.MoveBy (gameObject, iTween.Hash("y", 4.0f, "time", 8, "delay", 0.0, "looptype",
                iTween.LoopType.none, "oncomplete", "MoveComplete") );
}


private void ElevatorMoveDown()
{
        iTween.MoveBy (Elevator, iTween.Hash("z", 4.0f, "time", 8, "delay", 0.0, "looptype",
                iTween.LoopType.none) );
        iTween.MoveBy (gameObject, iTween.Hash("y", 4.0f, "time", 8, "delay", 0.0,
                "looptype", iTween.LoopType.none, "oncomplete", "MoveComplete") );
}

private void MoveComplete()
{
        ElevatorIsMoving = false;
        print("elevator moving end");
}
```

2.7 Laboratorium nr 7 – „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, ale przetwarzane przez podsystem grafiki danej aplikacji (łączenie z realizacją skryptów i obsługą fizyki).

Unity3D oferuje system Umbra, który realizuje optymalizację, jednakże wymagana jest wersja Pro środowiska Unity3D. Dla wersji Indie Unity3D można też zastosować pakiet SECTR VIS dostępny w Asset Store. Istnieją też darmowe rozwiązanie o nazwie M2H Culling, dostępne w Asset Store, dodatkowe informacje można odszukać pod adresem: http://www.m2h.nl/unity/. Autorem systemu M2H Culling jest Mike Hergaarden.

Głównym zadaniem tego laboratorium jest dodanie, w oparciu o poprzedni przykład prototypu gry z labiryntem, systemu optymalizującego proces wizualizacji labiryntu. W tym celu można wykorzystać pakiet M2H Culling.



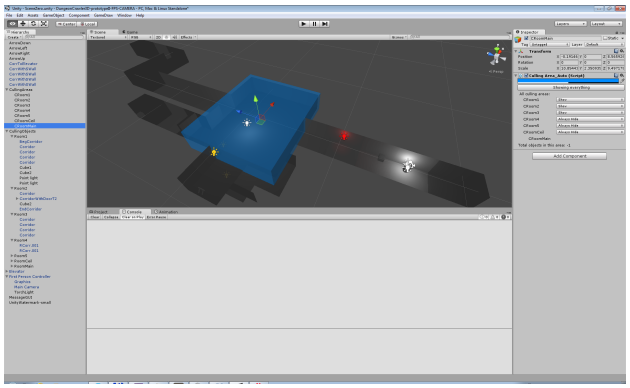
Rysunek 8: Podział elementów/obiektów na obszary, które będą ukrywane dynamicznie w zależności od pozycji gracza, pakiet M2H pozwala określać rożne kolory dla poszczególnych obiektów otaczających

2.8 Uwagi związane z zadaniem

Ogólnie stosując pakiet M2H Culling, należy utworzyć dwa obiekty GameObject np.: CullingObject oraz CullingAreas. 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 9: Ustalenie obszaru otaczającego dla głównego pomieszczenia

W drugim obiekcie CullingAreas znajdują się obiekty otaczające obszary stanowiące spójne obszary gry. 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.

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

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 Unity3D 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 i bądź ukrywane.

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

2.9 Laboratorium nr 8 – „Obsługa sieci – komunikator w Unity3D”

Głównym zadaniem tego laboratorium będzie zbudowanie nieskomplikowanym 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 serwer oraz klienta, powinna istnieć możliwość zamknięcia odłączenie się od serwera, czy ekrany klienta i powrót do okna startowego aplikacji.

2.10 Uwagi związane z zadaniem

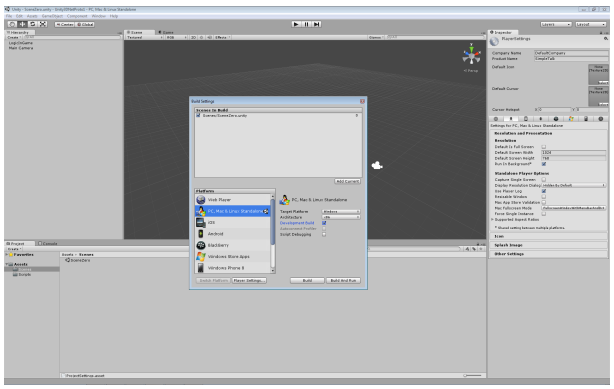
W najprostszej postaci komunikator wymaga obecności tylko obiektu kamery oraz obiektu np.: MainLogicObject, do którego zostanie podłączony skrypt do obsługi komunikatów oraz komponenty Network View.

Skrypt do obsługi komunikatora sieciowego może przyjąć następującą postać.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class NetworkCode : MonoBehaviour {
5      public string connectionIP = "127.0.0.1";
6      public int connectionPort = 25001;
7      public int maxConnections = 15;
8
9      private static ArrayList messages = new ArrayList();
10     private Vector2 scrollView = Vector2.zero;
11
12     private string message;
13
14     void Start () {
15         message = "" ;
16     }
17
18     void OnGUI() {
19         if (Network.peerType == NetworkPeerType.Disconnected) {
20             GUI.Label(new Rect(10, 10, 200, 20), "Status: Disconnected");
21             if (GUI.Button(new Rect(10, 30, 120, 20), "Client Connect")) {
22                 Network.Connect(connectionIP, connectionPort);
23             }
24             if (GUI.Button(new Rect(10, 50, 120, 20), "Initialize Server")) {
25                 Network.InitializeServer(maxConnections, connectionPort, false);
26             }
27         }
28
29         if (Network.peerType == NetworkPeerType.Server) {
30             GUI.Label(new Rect(10, 10, 200, 20), "Run as server");
31         }
32
33         if (Network.peerType == NetworkPeerType.Client) {
34             GUI.Label(new Rect(10, 10, 300, 20), "Status: Connected as Client");
35
36             if (GUI.Button(new Rect(10, 30, 120, 20), "Disconnect")) {
37                 Network.Disconnect(200);
38             }
```

```
39
40              message = GUI.TextField(new Rect(0, 100, 150, 25), message);
41              if (GUI.Button(new Rect(150, 200, 50, 25), "Send")) {
42                  networkView.RPC("ReciveMessage", RPCMode.All, message);
43                  message = "";
44              }
45          }
46          GUILayout.BeginArea(new Rect(0 , 120 , 400 , 200));
47          scrollView = GUILayout.BeginScrollView(scrollView);
48          foreach(string c in messages) {
49              GUILayout.Label(c);
50          }
51          GUILayout.EndArea();
52          GUILayout.EndScrollView();
53      }
54
55      [RPC]
56      private void ReciveMessage(string sentMess) {
57          messages.Add(sentMess);
58      }
59  }
```

Ponieważ na obecną chwilę nie można uruchomić dwóch kopii środowiska Unity3D,to należy zbudować wersję binarną aby przeprowadzić testy. Proces budowy jest wykonywany za pomocą opcji Build z menu File, Rys. 10. 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 10: Parametry budowy aplikacji w postaci binarnej

2.11 Laboratorium nr 9 – „Obsługa sieci – prototyp gry sieciowej dla kilkunastu graczy”

Zadanie realizowane na tym laboratorium stanowi rozwinięcie poprzedniego zadania z Lab. Nr 8. 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.

2.12 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 rozpoczynał 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 pewnej zmiany dotyczącej sposobu sterowania.

Skrypt odpowiadający za obsługę podstawowe logiki podłączony do Spawn point

```
using UnityEngine;
using System.Collections;

public class NetworkLogic : MonoBehaviour {

        public GameObject PlayerPrefab;

        public string connectionIP = "127.0.0.1";
        public int connectionPort = 25001;
        public int maxConnections = 15;

        public bool playerConnected;
        private int groupID = 1;


        void Start () {
                playerConnected = false;
        }

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

Niezbędne jest tez wniesienie poprawki do skryptu FPSInputController.js jaki znajduje się w pakiecie Character Controller. 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 zrealizowana 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:

```
1         void CreateNetworkPlayer(int playerType)
2         {
3                 playerConnected = true;
4                 var g = (GameObject)Network.Instantiate(PlayerPrefab,
5                         transform.position, transform.rotation, groupID);
6
7
8                 if(playerType == 0)
9                         g.AddComponent("NetworkPlayerLogicServer");
10
11                if(playerType == 1)
12                        g.AddComponent("NetworkPlayerLogicRemote");
13
14
15                var obj = g.GetComponentInChildren<Camera>();
16                obj.camera.enabled = true;
17                camera.enabled = false;
18        }
```

W przypadku serwera wymagana jest serializacja danych w następujący sposób:

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class NetworkPlayerLogicServer : MonoBehaviour {
5
6
7         void OnSerializeNetworkView( BitStream stream, NetworkMessageInfo info )
8         {
9                 Vector3 position = Vector3.zero;
10                Quaternion rotation = Quaternion.identity;
11
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 płynna 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     }
```

2.13 Laboratorium nr 10 – „Obsługa sieci – prototyp gry sieciowej – sterowanie autorytatywne”

Zadanie realizowane na tym laboratorium stanowi bezpośrednią kontynuację zadania z Lab. Nr 9. Celem, jest wbudowanie tzw. sterowania autorytatywnego. Polega ono na tym iż informacje o sterowaniu np.: wciśnięte klawisze są przekazywane do serwera który następnie przekazuje informacje do wszystkich klientów który 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.

2.14 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 rozpoczynał 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 są następujące:

```
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, NetworkMessageInfo info )
25 {
26         if( networkView.isMine )
27                 return;
28
29         if( !Network.isServer )
30                 return;
31
32         playerRemote.directionVector = dirVector;
33         playerRemote.inputJump = inJump;
34         playerRemote.SimulateInput();
35
36         if( Vector3.Distance( transform.position, position ) > 0.1f )
37         {
38                 networkView.RPC( "CorrectPlayerState", info.sender, transform.position );
39         }
40 }
```

2.15 Laboratorium nr 11 – „Animacja obiektów 3D”

2.16 Laboratorium nr 12 – „Obsługa shaderów”

2.17 Laboratorium nr 13 – Podłączenie do bazy danych z poziomu Unity3D

3 Dalsze informacje

Przegląd pozycji książkowych dotyczących środowiska Unity3D w języku angielskim można odszukać na stronie http://unity3dbooks.com/. Poniższy spis literatury odnosi się do wszystkich ćwiczeń i zadań z przedmiotu „Programowanie gier 3D”.

Literatura

[1] Materiały dostępne na stronie głównej Unity3D, http://www.unity3d.com.

[2] 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

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