„KOMPUTEROWA GRA 3D TYPU FPS Z ELEMENTAMI GRY RPG”

Praca inżynierska

Autor: Kamil Mateusz Morawiecki

28 STYCZNIA 2021

UNIWERSYTET PEDAGOGICZNY IM. KEN W KRAKOWIE
Instytut Informatyki

Spis treści

Wprowadzenie

Poprzez dynamiczny rozwój rynku gier komputerowych pod kątem finansowym i jakościowym spełniają one coraz to nowe zadania, również te z aspektu kultury. Przydzielane są nagrody pokroju „Oscarów" dla aktorów dubbingowych, nagrody dla najlepszych wizjonerów, słynne wyróżnienie „Game Of The Year - GOTY" i wiele innych. Dla specjalistów w dziedzinach programowania w środowiskach silników gier, grafików, specjalistów z dziedziny opraw audiowizualnych oraz osób innych specjalności pojawiają się obiecujące perspektywy zaistnienia na rynku. W związku z rozwojem silników gier pojawiły się również możliwości ich projektowania przez niezależnych twórców, tzw. Indie-Developer'ów. Chęć stworzenia własnego, unikatowego rozwiązania była przyczyną do powstania niniejszej pracy.

Cel pracy

Celem pracy było stworzenie funkcjonalności oraz elementów gry 3D typu pierwszoosobowej gry akcji (z ang. First Person Shooter) z elementami gry RPG (z ang. Role Playing Game), czyli gry o złożonej strukturze związanej z realizowaniem konkretnych zadań oraz o nieliniowej fabule, czyli takiej, w której każdy wybór gracza może powodować inną konsekwencję w przyszłości rozgrywki w zależności od podjętych decyzji. Przede wszystkim najważniejszym aspektem było stworzenie prostego systemu do implementowania dialogów w czasie rzeczywistym w czasie trwania zadania. Równie ważne było podejście wizualne. Tutaj również wzięto pod uwagę projektowanie wyglądu scen za pomocą własnej pracy twórcy jak i gotowych, darmowych rozwiązań.

Użyte technologie

Unity Engine w wersji 2019.3.5f1

Unity Engine jest silnikiem gier, który znajduje szerokie zastosowanie na rynku. Przede wszystkim jest wykorzystywany w produkcjach o niskim budżecie i stosunkowo małej jakości. Językiem programowania w obrębie tego środowiska jest C#. Opanowanie funkcjonalności silnika jest ułatwione w stosunku do konkurencyjnych silników, ponieważ w źródłach internetowych można znaleźć olbrzymią ilość materiałów dydaktycznych, dokumentacji, a w przypadku ograniczonej mocy zespołu warto spojrzeć na „Asset Store". Jest to sklep z gotowymi rozwiązaniami do zaimplementowania. Część z nich jest płatna, a część darmowa. Jeżeli twórcy nie umieścili własnej umowy licencyjnej, wówczas twórca powinien polegać wyłącznie na umowie licencyjnej Unity. Pozwala ona na maksymalny roczny przychód firmy na poziomie 100 000 dolarów bez opłacania licencji silnika. W kolejnych etapach istnieją progi licencyjne uwarunkowane głównie przychodem. Właśnie dlatego jest to tak wpływowy silnik na rynku. Pozwala on na szybki start na rynku, a kiedy już zespół osiągnie sukces nie musi się martwić o zawiłość zasad opłacania rachunku za korzystanie z silnika.

3DS Max

Jest to dosyć popularny edytor grafiki 3D. Z głównych zalet można wyróżnić przede wszystkim mnogość dostępnych narzędzi. Wymaga on więcej wiedzy niż darmowe odpowiedniki, ale też i więcej oferuje pod kątem renderingu, narzędzi transformacyjnych, animacji oraz różnych efektów cząsteczkowych. W pracy wykorzystano go głównie do zaimplementowania stworzonej wcześniej sceny poprzez eliminację elementów niepotrzebnych.

Blender

Prawdopodobnie najbardziej popularny darmowy edytor grafiki 3D. Przede wszystkim cechuje go prostota oraz szybkość działania pod kątem tworzenia siatki. Jest licencjonowany na licencji GPL (z ang. General Public License). Właśnie dlatego jest tak popularny. Ponadto daje on wiele możliwości, które są dostępne tylko w płatnych edytorach tego rodzaju. Blender jest bardzo przyjaznym środowiskiem do tworzenia obiektów, które mają zostać wyeksportowane do Unity. Przy poszukiwaniu rekomendowanego środowiska edycji 3D dla silników gier Blender jest najczęściej polecany przez twórców poradników do Unity.

Visual Studio 2019

Visual Studio jest domyślnym środowiskiem programistycznym w obrębie silnika Unity. Cechuje go przede wszystkim czytelny interfejs, podpowiedzi w czasie pisania kodu, które znacząco ułatwiają i przyspieszają pracę oraz oferują możliwość debugowania w rzeczywistym czasie pracy silnika.

Użyte gotowe rozwiązania (tzw. „Asset'y")

- FirstPerson AIO – jest to gotowy obiekt posiadający skrypty poruszania się bohatera, pasek wytrzymałości zużywający się podczas biegu oraz system skakania i kolizji. Nie użyto w projekcie standardowego asset'u Unity, ponieważ od pewnego czasu jest on uznany za zdeprecjonowany i nie zaleca się jego używania w przyszłości.
- UMA (Universal Multipurpose Avatar) – jest to najtrudniejszy do korzystania asset. Przede wszystkim wymaga on nauki własnego interfejsu i zasad działania. Postacie instancjonowane są dynamicznie, czyli nie ma możliwości stworzenia postaci, a następnie skopiowania ich definicji własnych w formie tzw. Prefab'u. Takie rozwiązanie skutkuje błędami. W efekcie nawet tworzenie animacji jest kłopotliwe. Każda postać ma unikatowe cechy wyglądu, takie jak: wiek, fryzura, kolor włosów, kolor skóry, rozmiar nosa, uszu, policzków, oczu, postury oraz wielu, wielu innych. Aby obiekt był tworzony według założeń projektanta należy zdefiniować dwa typy danych: recepty oraz szafy. Pierwsze odnoszą się do cech genetycznych, a drugie do ubioru.
- Modern Weapons Pack – jest to zbiór obiektów 3D, które reprezentują najpopularniejsze typy broni.
- Grass Flowers – zawiera wysokiej jakości obiekty reprezentujące trawy o charakterze kwiatowym.
- Urban American Assets – zawiera modele 3D pojazdów typowo charakterystycznych dla rynku Amerykańskiego.
- Conifiers [BOTD] – jest to zbiór drzew, z gotowym oskryptowaniem pod kątem LOD (level of Detail), losowości umieszczania drzew pod kątem ilości, wysokości i kształtu. Dodatek ten oparty jest na narzędziu SpeedTree. Również pozwala on na zaimplementowanie fizyki wiatru.

Własnoręcznie zaprojektowane rozwiązania

1.1. System strzelania – Aby gra mogła spełniać warunek pierwszoosobowej gry akcji zaimplementowano własne rozwiązanie, które miało symulować strzelanie z broni palnej. System jest prosty w swojej zasadzie działania. Gracz klikając w lewy przycisk myszy uaktywnia skrypt oddania strzału. Podczas aktywacji strzału wyświetlany jest przez ułamek sekundy obiekt imitujący rozbłysk po wystrzeleniu naboju. Uwzględniona jest również precyzja strzału uwarunkowana losowym odchyłem, który zależy od parametru celności podawanej w procentach. W przypadku strzału przez gracza z parametrem 100% celności zawsze trafi on w odpowiedni punkt.

```
RaycastHit hit;
rX = Random.Range(-offset, offset);
rY = Random.Range(-offset, offset);

if (Physics.Raycast(playerCam.transform.position, playerCam.transform.forward + new Vector3(rX, rY, 0.0f), out hit, range))
{
```

Ilustracja 1 - Fragment kodu odpowiedzialny za zmniejszenie precyzji strzelania
Źródło: opracowanie własne

W przypadku 0% celności strzał może się odbyć w losowym ciągu prostej z zakresu od -100 stopni do 100 stopni po szerokości i wysokości. Dzieje się tak, ponieważ podczas instancjonowania strzału dodawany jest kolejny obiekt typu „Vector3” z parametrami losowego przesunięcia linii strzału do wektora strzału. Na Ilistracji 1 zobrazowano konkretny fragment kodu źródłowego odpowiedzialny za losowe obniżenie celności. De facto można wyobrazić go sobie jako stożek projekcyjny. Im bliżej gracz znajduje się do celu tym łatwiej mu go będzie trafić. Również jeżeli wirtualny pocisk znajdzie się na drodze obiektu typu NPC, czyli bohatera niezależnego, innej postaci niż postać sterowana przez gracza, wówczas zostaną mu zadane obrażenia i zainstancjonowany zostanie obiekt typu „bullet hole”(dziura po kuli) pod kątem strzału w stosunku do tzw. Collider’a (obiektu, który ma za zadanie symulować nieprzenikalność obiektów i zachować ich kolizyjność poprzez zastosowanie symulacji zaimplementowanych w silniku). Dziura po kuli jest niczym innym jak trójwymiarowym obiektem typu „Plane” z nałożoną teksturą i poziomem przezroczystości. Zastosowano dwa typy dziur po kulach: organiczny, który ma za zadanie być instancjonowany w przypadku strzału w istotę żywą, a drugi nieorganiczny ma się pojawić w przypadku strzału w obiekt martwej natury sceny. Jest jeszcze trzecia możliwość, a mianowicie zniszczenie obiektu poprzez strzał. Zastosowano to w przypadku obiektów o tagu „Destroyable”. Przykładem są szkła w oknach budynków. Omawiany kod do strzelania zaimplementowano w rozdziale „Kody źródłowe” w sekcji „CameraShooter.cs”.

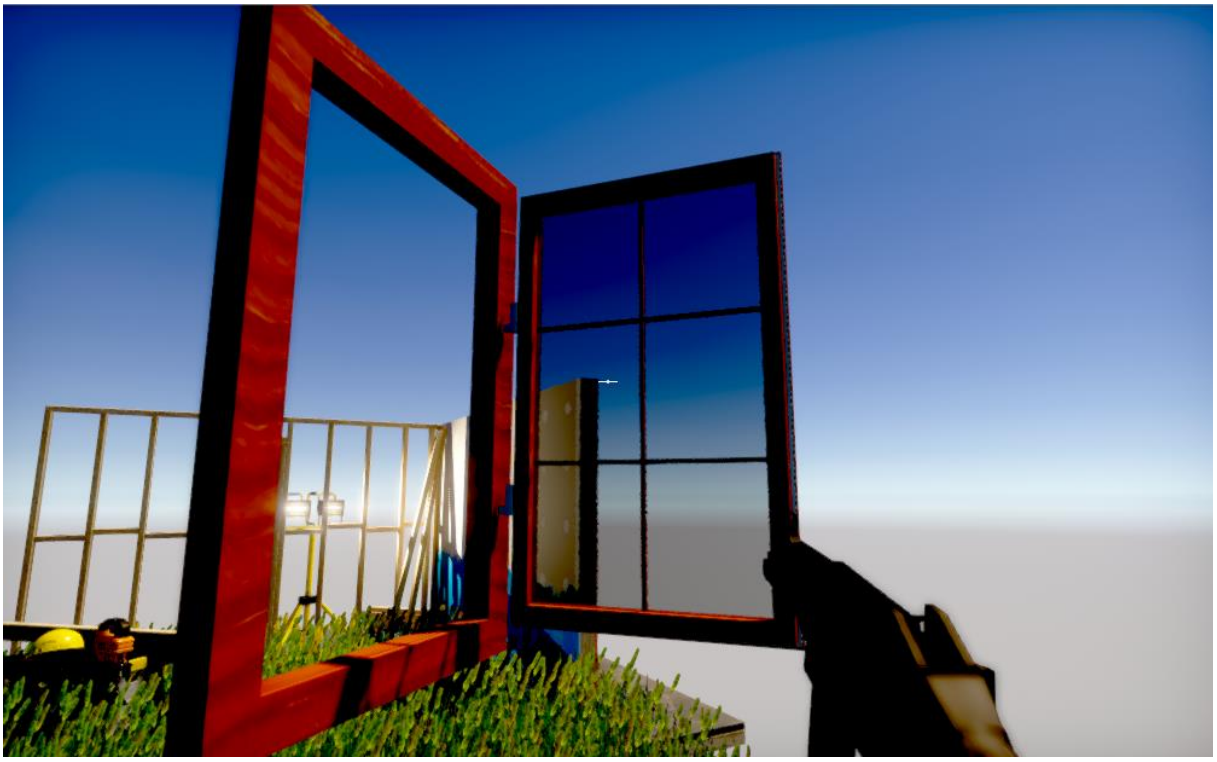
Ilustracja 2 - Przykładowy przebieg zadania
Źródło: Opracowanie własne

1.2. System zadań i dialogów – Aby projektant zadań (Quest Designer) otrzymał narzędzie proste i zrozumiałe należy ułatwić mu pracę poprzez stworzenie metod wykonujących dane czynności związane z zadaniami i dialogami. System zadań może

być bardzo złożony, więc nie skupiono się na realizowaniu jego metod, a zmiennych bazowych typu bool. Każde zadanie (quest) ma cechować się posiadaniem tychże zmiennych, a są one następujące:

- isQuestOptional – jeżeli zmienna ma wartość ‘false’, to wówczas zadanie jest obowiązkowe do wykonania aby ukończyć grę. W innym przypadku jest ono opcjonalne.
- isQuestAvailable – jeżeli zmienna ma wartość ‘true’, to wówczas zadanie może się pojawić w przyszłości. Powodem zaimplementowania tej zmiennej jest charakterystyka gry RPG, a konkretnie różność zadań pod względem podjętych decyzji. Oczywistym jest, że należy wyeliminować pojawianie się zadań, które nie będą miały sensu w danym scenariuszu wyborów gracza.
- isQuestReadyToActive – jeżeli zmienna ma wartość ‘true’, to wówczas zadanie może zostać rozpoczęte. W przeciwnym wypadku gracz nawet nie dostrzeże możliwości jego rozpoczęcia.
- isQuestActive – w przypadku, gdy zadanie jest już rozpoczęte i jest w toku, wartość tej zmiennej będzie równa ‘true’.
- isQuestFinished – w przypadku, gdy zadanie zostanie zakończone, wartość tej zmiennej będzie równa ‘true’. Nie daje ona informacji o zakończeniu zadania z sukcesem.
- isQuestFailure – w przypadku, gdy wartość zmiennej isQuestFinished będzie równe ‘true’, a omawiana zmienna będzie miała również wartość true, to wówczas zadanie zostanie sklasyfikowane jako „zakończone niepowodzeniem”. Jeżeli wartość tej zmiennej będzie równa ‘false’, a zmienna isQuestFinished będzie miała wartość ‘true’, to wówczas zadanie można sklasyfikować jako „zakończone z powodzeniem”.

W obrębie konkretnego zadania, które musi być skryptem C# przyszły projektant powinien realizować system dialogów poprzez wysyłanie informacji o ich przebiegu w dwóch formach: podpisowej, która określa monolog rozmówcy i wysyłania dostępnych opcji dialogowych po danym monologu. Służą do tego odpowiednio metody: `sendSubtitle(string monolog_rozmowcy) oraz setDialogueOption(int id_opcji, string tekst_opcji)`. Menedżer dialogów podczas wyboru decyzji nasłuchuje klawisza „Enter” i wysyła z powrotem do skryptu zadania wybraną opcję dialogową. Skrypt ten musi być zatem być pisany przez w miarę doświadczonego programistę, który ma już stworzony pewien system rozproszonej ekspozycji danych.

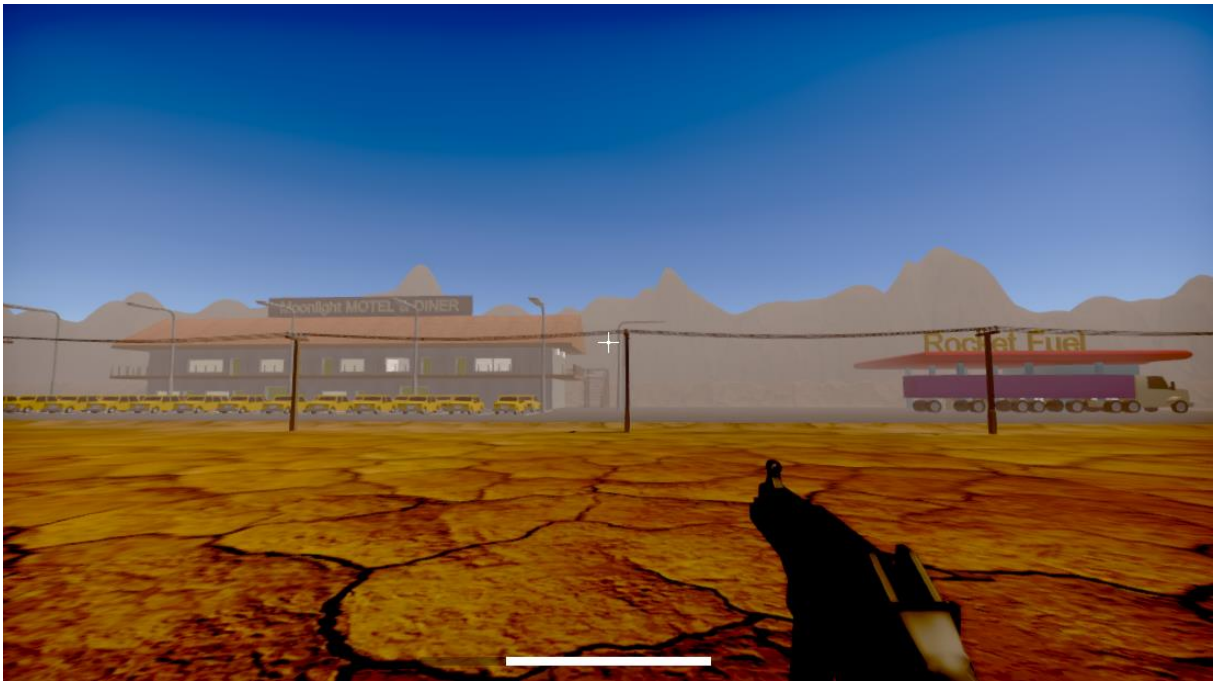


Ilustracja 3 - Przykładowe zastosowanie interakcji z oknem
Źródło: Opracowanie własne

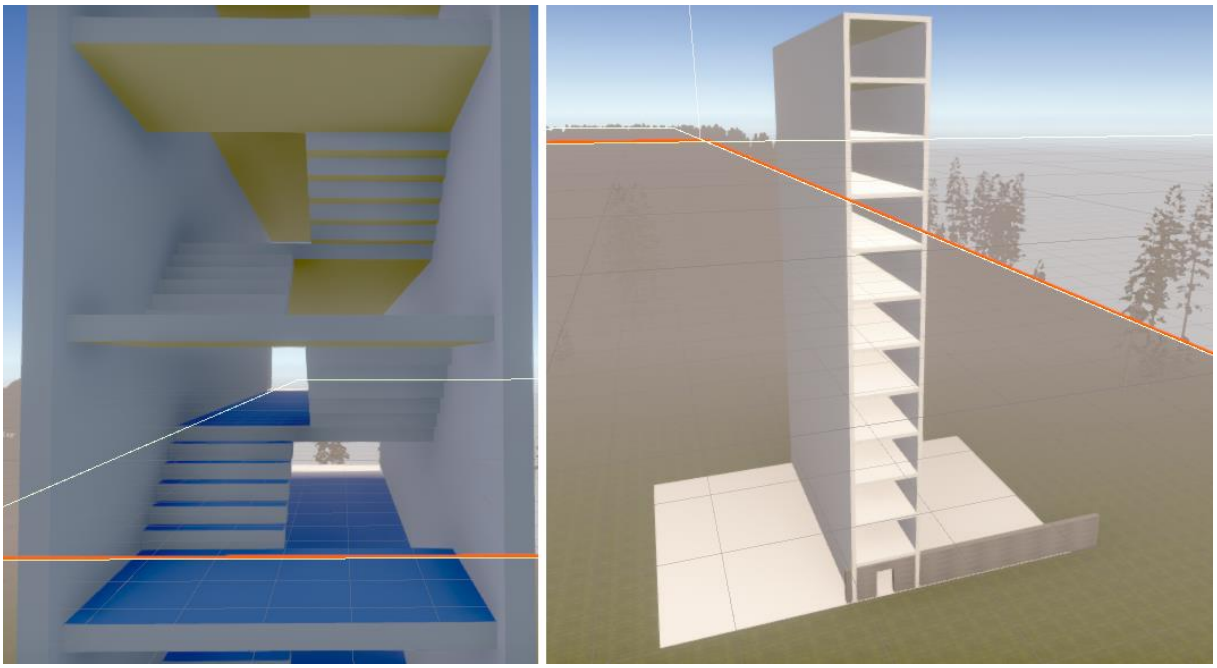
1.3. System interakcji z obiektami – jest to proste rozwiązanie, które pozwala na interakcję z obiektami, które posiadają taką możliwość. W przypadku, gdy gracz znajduje się blisko interaktywnego obiektu, wówczas może on uruchomić animację przypisaną do niego po kliknięciu przycisku „E” z klawiatury. Zasada działania jest podobna i względnie ta sama dla obiektów typu okna i drzwi.

1.4. Świat gry – modele 3D. W sumie spędzono ponad 200 godzin nad stworzeniem jednej z głównych lokacji, mianowicie stacji benzynowej i motelu, które to należą w świecie gry do frakcji „Nomadów”. Na ilustracji 4 zobrazowano budynki z zewnątrz, a na ilustracjach 6 i 7 zobrazowano wynik pracy wewnątrz tychże budynków. W czasie pracy nad projektem inżynierskim na nowo stworzono obiekty 3D w środowisku Blender, a były to:

- krzesła, meble i stoły,
- Szkielet budynku z uniwersalnymi segmentami klatki schodowej (ilustracja 5),
- Okna i drzwi,
- Wyposażenie wnętrz: komputery, telewizory i kolumna głośnikowa,

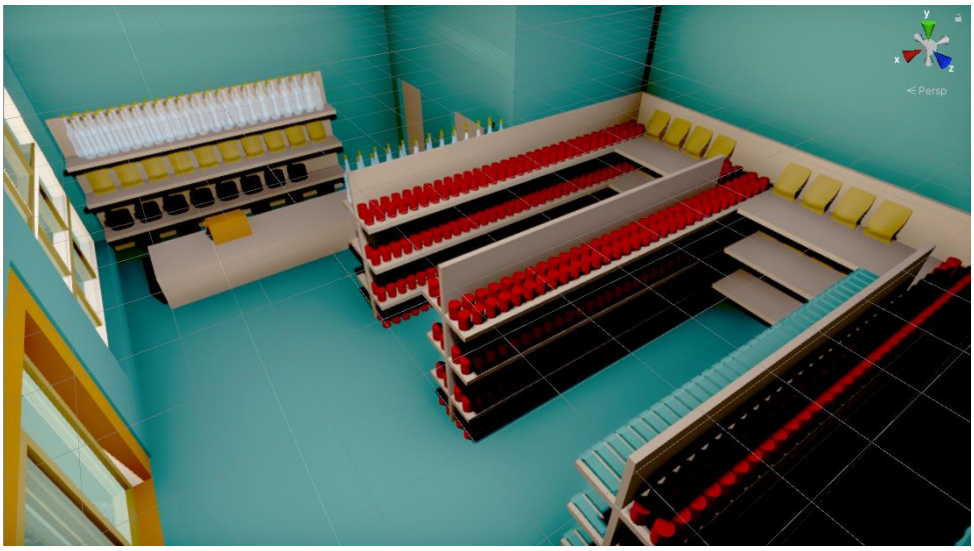


Ilustracja 4 - Część mapy składająca się z dużej liczby elementów
Źródło: Opracowanie własne



Ilustracja 5 - Szkielet uniwersalny bloku mieszkalnego
Źródło: Opracowanie własne

Ilustracja 6 - Przykładowe wnętrze motelu
Źródło: Opracowanie własne



Ilustracja 7 - Przykładowy wygląd wnętrza stacji benzynowej
Źródło: Opracowanie własne

Koncepcja fabuły i świata gry

Fabuła gry

Gra ma za zadanie przedstawiać fantastyczny obraz rzeczywistości i wydarzenia w post nuklearnym świecie. Główny bohater będzie podróżował do celu, który zostanie podany na początku gry. W trakcie swej podróży napotka na problem „utknięcia” w lokacji z której można wyjechać jedynie korzystając z dwóch dróg. W trakcie wejścia w rozdział związany z tą historią bohater przyjedzie do tejże lokacji nie mając świadomości, że nie będzie mógł już jej opuścić z powodu obaw służb państwowych przed odpowiedzialnością za wyciek radioaktywny w tym rejonie. Z tegoż powodu każdy kto znalazł się w tejże lokacji może ją opuścić jedynie za pozwoleniem władz. Przynajmniej teoretycznie, bo skoro jest to gra RPG, to istnieje wiele możliwości zakończenia tej historii. W grze zaistnieje podział na prolog, w którym gracz będzie mógł się bliżej zapoznać z głównym bohaterem

i z jego historią. Również takie wprowadzenie pozwoli lepiej poznać mechanikę rozgrywki. Rozdział 1 będzie obejmował właśnie tą skażoną radioaktywnie okolicę, której nie można opuścić. Uniwersum będzie składało się z 3 części, pierwsza obejmie jedynie rozbudowany rozdział 1 z krótkim prologiem. Kolejne części nie są częścią tego projektu. Schemat fabularny zamieszczono na diagramie 1.

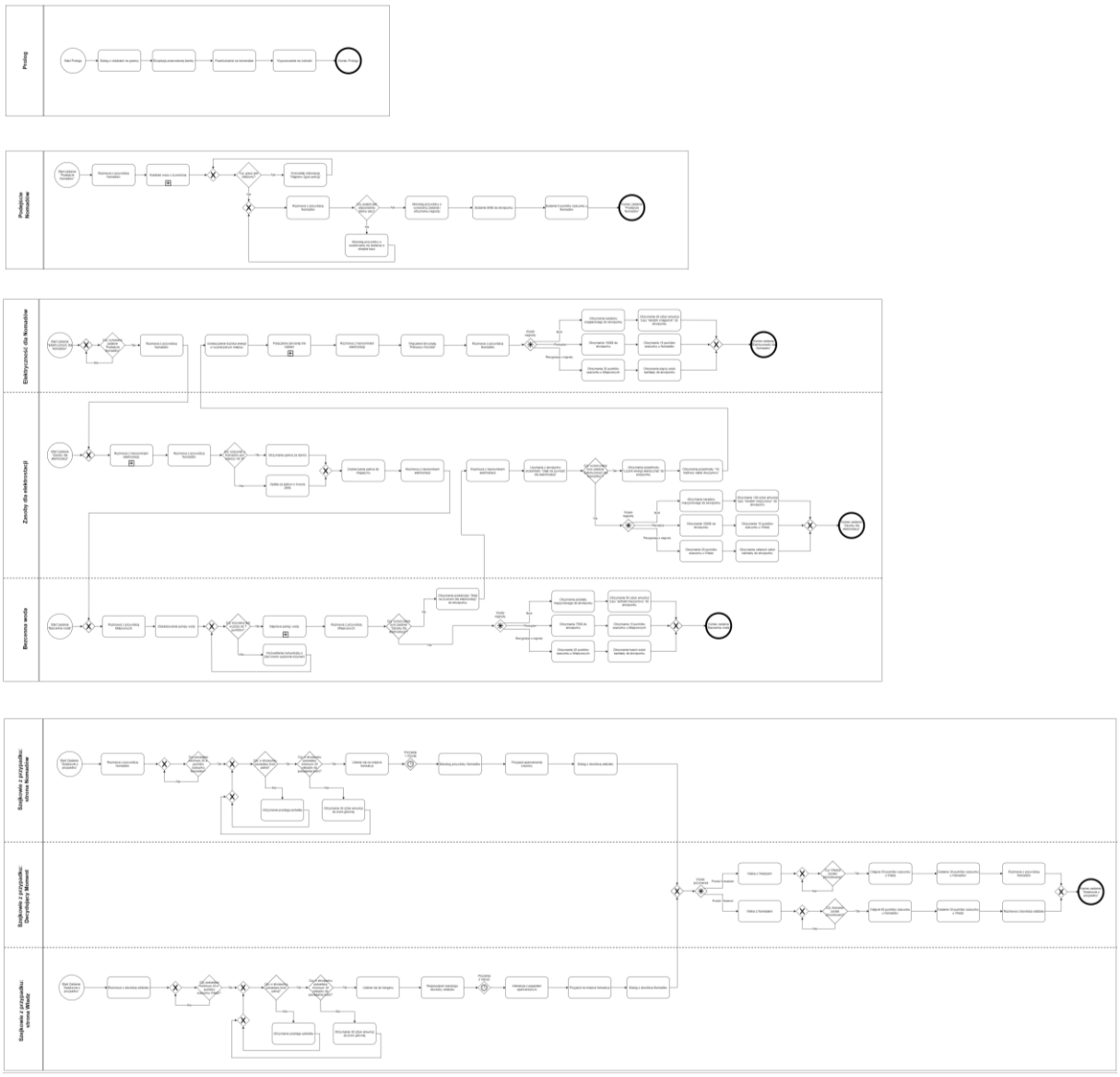
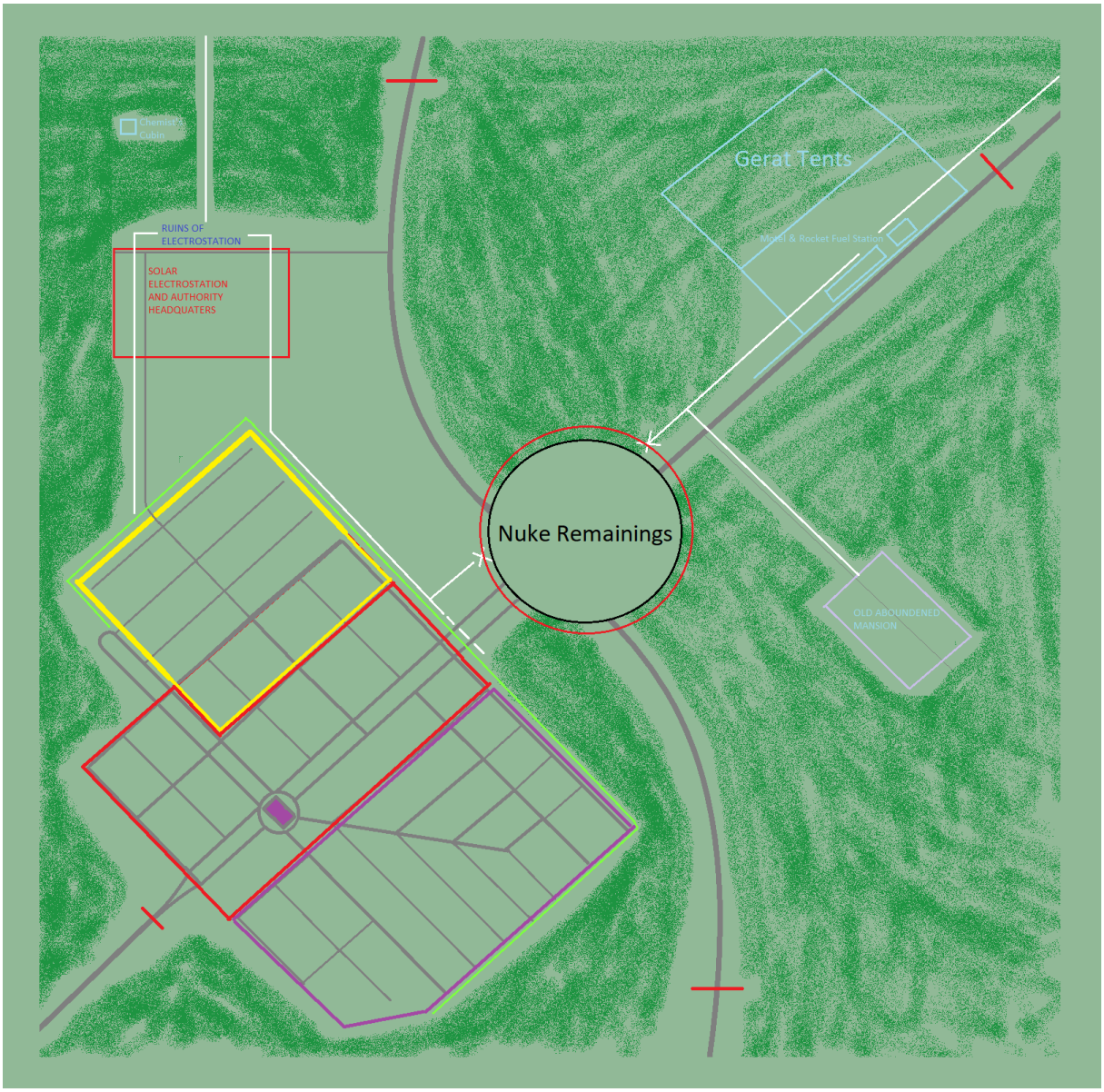


Diagram 1 - przebieg fabuły gry
Źródło: opracowanie własne

Mapa gry

W założeniu, aby gra miała sensowny podział lokacji, ustalono przynależność poszczególnych frakcji (grup postaci) na poszczególnych terenach do nich należących. Na Mapie 1 kolorem czerwonym zaznaczono terytoria Władz (Authorities), fioletowy to kolor terytoriów Lokalnych Mieszkańców (Townies), żółty to kolor frakcji Blokersów (Hoodies), a niebieski to terytoria Nomadów (Nomads). W centralnym punkcie mapy widnieje krater po eksplozji jądrowej, który został odseparowany od otoczenia przez władze i wstęp jest tam surowo wzbroniony. Pod ich jurysdykcję podlega również terytorium Elektrowni (Electrostation) i centrum miasta. W przypadku wejścia przez gracza na terytorium danej frakcji podczas posiadania ujemnych punktów reputacji

u danej z nich, istnieje możliwość zaatakowania gracza w zależności od stopnia negatywnej reputacji. Czasami może się okazać, że dane zadanie z poziomu obiektu ‘QuestManager’ zostanie pozbawione możliwości ukończenia bądź nawet rozpoczęcia dopóki reputacja postaci głównej nie wzrośnie w danej frakcji.



Mapa 1 - podstawowa koncepcja podziału terytoriów gry na każdą z frackji
Źródło: Opracowanie Własne

Kody źródłowe

[QuestManager.cs]

```
public class QuestManager : MonoBehaviour{
    struct QuestParameters
    {
        GameObject questActivationObject;

        bool isQuestOptional;
        bool isQuestAvailable;
        bool isQuestReadyToActive;
        bool isQuestActive;
```

```
        bool isQuestSuccess;
        bool isQuestFailure;
    }

    public GameObject[] quests;
}
```

[SampleQuest1.cs]

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;


public class SampleQuest1 : MonoBehaviour

{

   //must be automatic in future

   public GameObject player;

   public GameObject questFinishObject;

   public GameObject dialogueManager;


   float distX = 500.0f;

   float distY = 500.0f;

   float distZ = 500.0f;


   float distX_starter;

   float distY_starter;

   float distZ_starter;


   float distX_finisher;

   float distY_finisher;

   float distZ_finisher;


   //some variables for purposes of this excact task;

   bool goneAwayFromEmployer;

   bool wentToFinishingObject;
```

```
float interactionDistance;

//must consider to use connection from QuestManager's definition of this struct
struct QuestParameters
{
    public GameObject questActivationObject;

    public bool isQuestOptional;
    public bool isQuestAvailable;
    public bool isQuestReadyToActive;
    public bool isQuestActive;
    public bool isQuestSuccess;
    public bool isQuestFailure;
}

QuestParameters qp;
public GameObject questActivationObject;

//table for storing dialogue sequences, last is always 1, if there is a 0 somewere in the
middle or beginning it means, that there was some negative attitude
private bool[] dialogueSequences;

//variable for storin last dialogue decision
private int lastDecision;

void Start()
{
    //in future this could 'come' from QuestManager
```

```
    qp.questActivationObject = questActivationObject;

    qp.isQuestOptional = false;
    qp.isQuestAvailable = true;
    qp.isQuestReadyToActive = true;
    qp.isQuestActive = false;
    qp.isQuestSuccess = false;
    qp.isQuestFailure = false;

    interactionDistance = 2.0f;
    goneAwayFromEmployer = false;
    wentToFinishingObject = false;

    if (qp.isQuestAvailable && qp.isQuestReadyToActive)
    {
        Debug.Log("Quest just ready to active");
    }

    //we will need to declare specific number of dialogue sequences during this quest
    dialogueSequences = new bool[10];
}

IEnumerator waitCoroutine(int s)
{
    yield return new WaitForSeconds(s);

    dialogueManager.GetComponent<DialogueManager>().deactivateSubtitles();
}
```

```
// Update is called once per frame
void Update()
{
    if (!(qp.isQuestSuccess))
    {
        if (!(qp.isQuestFailure))
        {
            if (!(qp.isQuestActive))
            {
                distX = Mathf.Abs(player.transform.position.x -
questActivationObject.transform.position.x);
                distY = Mathf.Abs(player.transform.position.y -
questActivationObject.transform.position.y);
                distZ = Mathf.Abs(player.transform.position.z -
questActivationObject.transform.position.z);



                if ((distX < interactionDistance) && (distY < interactionDistance) && (distZ <
interactionDistance))
                {
                    qp.isQuestActive = true;
                    Debug.Log("Quest Activated");


                    dialogueManager.GetComponent<DialogueManager>().activateSubtitles();

dialogueManager.GetComponent<DialogueManager>().sendSubtitle("Welcome, stranger. I
think even such a noob like you can do something for me. Besides, what the hell are you
doing at this land?");


                    //activating dialogue
                    dialogueManager.GetComponent<DialogueManager>().activateDialogue();
```

```
            //clearing all dialogue options
            dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();

            //setting dialogue options
            dialogueManager.GetComponent<DialogueManager>().setDialogueOption(0,
"Nothing. I just got here.");
            dialogueManager.GetComponent<DialogueManager>().setDialogueOption(1,
"What's your problem?");

            //starting first (0) dialogue sequence
            dialogueSequences[0] = true;
        }
    }

    if (qp.isQuestActive && dialogueSequences[0] && !dialogueSequences[1])
    {
        if (Input.GetKeyDown(KeyCode.Return))
        {
            lastDecision =
dialogueManager.GetComponent<DialogueManager>().getDecision();

            if (lastDecision == 1)
            {
                dialogueSequences[1] = false;
                dialogueManager.GetComponent<DialogueManager>().sendSubtitle("I am
very dissapointed with your attitude. I will not give You any job anymore.");//here instead of
Debug.Log should be coorutine for handling response monologue of interlocutor(person,
wchich we are just talkink to)
            }
            else if (lastDecision == 0)
            {
```

```
                dialogueSequences[1] = true;

                dialogueManager.GetComponent<DialogueManager>().sendSubtitle("OK, If
You find a guy who ownes me some money, please tell him that Big Beard is sending
regards.");//here instead of Debug.Log should be coorutine for handling response
monologue of interlocutor(person, wchich we are just talkink to)

            }


            StartCoroutine(waitCoroutine(5));

        }

    }


    if (qp.isQuestActive && dialogueSequences[1] && !dialogueSequences[2])

    {

        //ending dialogue

        dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();


        //clearing all dialogue options

        dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();


        distX_finisher = Mathf.Abs(player.transform.position.x -
questFinishObject.transform.position.x);

        distY_finisher = Mathf.Abs(player.transform.position.y -
questFinishObject.transform.position.y);

        distZ_finisher = Mathf.Abs(player.transform.position.z -
questFinishObject.transform.position.z);


        distX_starter = Mathf.Abs(player.transform.position.x -
questActivationObject.transform.position.x);

        distY_starter = Mathf.Abs(player.transform.position.y -
questActivationObject.transform.position.y);

        distZ_starter = Mathf.Abs(player.transform.position.z -
questActivationObject.transform.position.z);
```

```
            if (!(goneAwayFromEmployer) && (distX_starter > interactionDistance + 5.0f) &&
(distY_starter > interactionDistance + 5.0f))
            {
               goneAwayFromEmployer = true;
               Debug.Log("I went away from employer");

               dialogueManager.GetComponent<DialogueManager>().deactivateSubtitles();//
dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();//This all
should be with coorutine while listening to MPC's monologue
            }

            if ((distX_finisher < interactionDistance) && (distY_finisher < interactionDistance)
&& (distZ_finisher < interactionDistance))
            {
               wentToFinishingObject = true;

               dialogueManager.GetComponent<DialogueManager>().activateSubtitles();
               dialogueManager.GetComponent<DialogueManager>().sendSubtitle("What do
you want, stranger?");

               //activating dialogue
               dialogueManager.GetComponent<DialogueManager>().activateDialogue();

               //clearing all dialogue options
               dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();

               //setting dialogue options
```

```
            dialogueManager.GetComponent<DialogueManager>().setDialogueOption(0,
"Big Beard is sending regards. Give me what you own to him.");

            dialogueManager.GetComponent<DialogueManager>().setDialogueOption(1,
"Hey thief, Big Beard send me to get what You've taken from him.");


            dialogueSequences[2] = true;


            Debug.Log("Quest Progress");
        }
    }


    if (qp.isQuestActive && dialogueSequences[2] && !dialogueSequences[3])
    {
        if (Input.GetKeyDown(KeyCode.Return))
        {
            lastDecision =
dialogueManager.GetComponent<DialogueManager>().getDecision();


            if (lastDecision == 0)
            {
                dialogueSequences[3] = true;
                dialogueManager.GetComponent<DialogueManager>().sendSubtitle("OK,
here You have his artefact. I don' need it anymore.");//here instead of Debug.Log should be
coorutine for handling response monologue of interlocutor(person, wchich we are just
talkink to)


                //ending dialogue
                dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();


                //clearing all dialogue options

dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();
```

```
            StartCoroutine(waitCoroutine(5));


//dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();
        }
        else if (lastDecision == 1)
        {
            dialogueSequences[3] = true;
            dialogueManager.GetComponent<DialogueManager>().sendSubtitle("OK,
here You have his artefact. I don' need it anymore. But, you were very rude to me, so I will
remember that.");//here instead of Debug.Log should be coorutine for handling response
monologue of interlocutor(person, wchich we are just talkink to) //and there sould come
some reputation mark--

            //ending dialogue
            dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();

            //clearing all dialogue options
dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();

            StartCoroutine(waitCoroutine(8));
        }

        //ending dialogue
        dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();

        //clearing all dialogue options
        dialogueManager.GetComponent<DialogueManager>().clearDialogueOptions();
```

```
            // dialogueManager.GetComponent<DialogueManager>().deactivateSubtitles();
        }
    }

    if (qp.isQuestActive && dialogueSequences[3])
    {
        distX_finisher = Mathf.Abs(player.transform.position.x -
questFinishObject.transform.position.x);
        distY_finisher = Mathf.Abs(player.transform.position.y -
questFinishObject.transform.position.y);
        distZ_finisher = Mathf.Abs(player.transform.position.z -
questFinishObject.transform.position.z);

        distX_starter = Mathf.Abs(player.transform.position.x -
questActivationObject.transform.position.x);
        distY_starter = Mathf.Abs(player.transform.position.y -
questActivationObject.transform.position.y);
        distZ_starter = Mathf.Abs(player.transform.position.z -
questActivationObject.transform.position.z);

        if ((distX_starter < interactionDistance) && (distY_starter < interactionDistance)
&& (distZ_starter < interactionDistance))
        {
            dialogueManager.GetComponent<DialogueManager>().activateSubtitles();

            dialogueManager.GetComponent<DialogueManager>().sendSubtitle("Good
Job! Here's your reward.");

            Debug.Log("Good Job! Here's your reward.");

            dialogueManager.GetComponent<DialogueManager>().deactivateDialogue();
```

```
                StartCoroutine(waitCoroutine(5));

                //here should come some income for propper quest ending
            }

        }
    }
  }
}
```

[OpenClose.cs]

```
public class OpenClose : MonoBehaviour
{
    public float minDistance = 1.5f;
    public float speed = 10.0f;
    public Animator animator;

    private GameObject window;
    private GameObject player;
    private bool isOpened = false;
    private bool rotateLeft = false;
    private bool rotateRight = false;


    // Start is called before the first frame update
    void Start()
    {
        window = this.gameObject.transform.GetChild(1).gameObject;
        player = GameObject.Find("FirstPerson-AIO");
    }

    // Update is called once per frame
    void Update()
    {
        if (countDistanceFromPlayer() <= minDistance)
        {
            //set interaction in UI as visiable

            if(Input.GetKeyDown(KeyCode.E))
            {
                if (isOpened == false)
                {
                    isOpened = true;
                    Debug.Log("Door Opened");
                    //window.transform.rotation = new
Quaternion(window.transform.rotation.x, window.transform.rotation.y,
window.transform.rotation.z + 1.0f, 1.0f);
                    StartCoroutine(Open());
                }
                else
                {
                    isOpened = false;
                    Debug.Log("Door Closed");
                    //window.transform.rotation = new
Quaternion(window.transform.rotation.x, window.transform.rotation.y,
window.transform.rotation.z - 1.0f, 1.0f);
                    StartCoroutine(Close());
                }

            }
        }


    }

    IEnumerator Open()
    {
        animator.SetBool("isOpening", true);

        yield return new WaitForSeconds(0.1f);

        animator.SetBool("isOpening", false);
```

```
    }

    IEnumerator Close()
    {
        animator.SetBool("isClosing", true);

        yield return new WaitForSeconds(0.1f);

        animator.SetBool("isClosing", false);
    }

    private float countDistanceFromPlayer()
    {
        return Vector3.Distance(window.transform.position, player.transform.position);
    }
}
```

[CharacterStats.cs]

```
public class CharacterStats : MonoBehaviour
{
    public int HP = 100;
    public float aimAccurency = 65.0f;
    private bool isAlive = true;

    // Update is called once per frame
    void Update()
    {
        if(HP <= 0 && isAlive)
        {
            Debug.Log("Death of " + this.gameObject.name);
            isAlive = false;
        }
    }

    public void takeDamage(int dmg)
    {
        HP -= dmg;
    }
}
```

[DialogueManager.cs]

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

public class DialogueManager : MonoBehaviour
{
    public GameObject canvas;
    public GameObject player;
    private bool dialogueActive;
    private int currentOption;
    private int maxOption;
    private Transform pointer;
    private GameObject subtitlesPanel;
    private GameObject subtitles;

    // Start is called before the first frame update
    void Start()
```

```
    {
        dialogueActive = false;
        currentOption = 0;
        maxOption = 0;

        pointer = canvas.transform.GetChild(0).gameObject.transform.GetChild(8);
        subtitlesPanel = canvas.transform.GetChild(1).gameObject;
        subtitles = subtitlesPanel.transform.GetChild(0).gameObject;
    }

    // Update is called once per frame
    void Update()
    {
        if(dialogueActive)
        {
            if ((Input.GetKeyDown(KeyCode.UpArrow) || Input.GetKeyDown(KeyCode.W)) &&
(currentOption > 0))
            {
                currentOption--;
                Debug.Log(currentOption);
                movePointerUp();
            }

            if ((Input.GetKeyDown(KeyCode.DownArrow) || Input.GetKeyDown(KeyCode.S))
&& (currentOption < maxOption))
            {
                currentOption++;
                Debug.Log(currentOption);
                movePointerDown();
            }
        }
    }

    private void movePointerUp()
    {
        pointer.position += new Vector3(0, 40.0f, 0);
    }

    private void movePointerDown()
    {
        pointer.position += new Vector3(0, -40.0f, 0);
    }

    public void activateSubtitles()
    {
        subtitlesPanel.active = true;
    }

    public void deactivateSubtitles()
    {
        subtitlesPanel.active = false;
    }

    public void sendSubtitle(string txt)
    {
        subtitles.GetComponent<Text>().text = txt;
    }

    public int getDecision()
    {
        return currentOption;
    }
```

```
    public void setNumberOfOptions(int num)
    {
        maxOption = num;
    }

    public void activateDialogue()
    {
        dialogueActive = true;
        canvas.transform.GetChild(0).gameObject.active = true;
        player.GetComponent<FirstPersonAIO>().enableCameraMovement = false;
        player.GetComponent<FirstPersonAIO>().playerCanMove = false;
    }

    public void deactivateDialogue()
    {
        dialogueActive = false;
        canvas.transform.GetChild(0).gameObject.active = false;
        player.GetComponent<FirstPersonAIO>().enableCameraMovement = true;
        player.GetComponent<FirstPersonAIO>().playerCanMove = true;

        currentOption = 0;
        maxOption = 0;
    }

    public void setDialogueOption(int x, string txt)
    {
        if (maxOption + 1 == x)
        {
            maxOption++;

canvas.transform.GetChild(0).gameObject.transform.GetChild(x).GetComponent<Text>().tex
t = txt;
        }
        else if (maxOption + 1 < x)
        {
            maxOption++;

canvas.transform.GetChild(0).gameObject.transform.GetChild(maxOption).GetComponent<Tex
t>().text = txt;
            Debug.Log("It's not allowed do declare number of dialogue option bigger
than max option value plus one. It was declared as maxOption + 1 instead.");
        }
        else if (x < 0)
        {

canvas.transform.GetChild(0).gameObject.transform.GetChild(x).GetComponent<Text>().tex
t = txt;
            Debug.Log("It's not allowed do declare number of dialogue option smaller
than 0. It was declared as 0.");
        }
        else

canvas.transform.GetChild(0).gameObject.transform.GetChild(x).GetComponent<Text>().tex
t = txt;
    }

    public void clearDialogueOption(int x)
    {

canvas.transform.GetChild(0).gameObject.transform.GetChild(x).GetComponent<Text>().tex
t = "";
```

```
    }

    public void clearDialogueOptions()
    {
        for (int i = 0; i < 8; i++)

canvas.transform.GetChild(0).gameObject.transform.GetChild(i).GetComponent<Text>().tex
t = "";

        currentOption = 0;
        maxOption = 0;
    }
}
```

[CameraShooter.cs]

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

public class CameraShooter : MonoBehaviour
{
    public float range = 50.0f;
    public int dmg = 20;
    public Camera playerCam;
    public float accurecy = 90.0f;
    public GameObject bulletHoleNonOrganic;
    public GameObject bulletHoleOrganic;
    public GameObject fireShoot;
    private float offset;
    float rX;
    float rY;

    // Start is called before the first frame update
    void Start()
    {
        offset = (100.0f - accurecy) * 0.001f;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            StartCoroutine(Shoot());


            RaycastHit hit;
            rX = Random.Range(-offset, offset);
            rY = Random.Range(-offset, offset);

            if (Physics.Raycast(playerCam.transform.position,
playerCam.transform.forward + new Vector3(rX, rY, 0.0f), out hit, range))
            {
                Debug.Log(hit.transform.name);//here should be method which will be
giving damage to object that player is shooting

                if (hit.transform.CompareTag("NPC_indie") ||
hit.transform.CompareTag("NPC_nomad") || hit.transform.CompareTag("NPC_panther") ||
hit.transform.CompareTag("NPC_townie"))
                {
                    Debug.Log("Thats's NPC!!!!");
```

```
hit.transform.gameObject.GetComponent<CharacterStats>().takeDamage(dmg);
                    Instantiate(bulletHoleOrganic, hit.point,
Quaternion.FromToRotation(transform.up, hit.normal));

                }
                else if (hit.transform.CompareTag("Destroyable"))
                {
                    Destroy(hit.transform.gameObject);
                }
                else
                {
                    Instantiate(bulletHoleNonOrganic, hit.point,
Quaternion.FromToRotation(transform.up, hit.normal));

                }
            }
        }

    }

    IEnumerator Shoot()
    {
        fireShoot.transform.Rotate(fireShoot.transform.rotation.x, 30.0f,
fireShoot.transform.rotation.z);
        fireShoot.SetActive(true);
        yield return new WaitForSeconds(0.05f);
        fireShoot.SetActive(false);
    }
}
```

Bibliografia

https://unity.com/support-services

https://forum.unity.com/threads/conifers-botd.645937/

https://docs.unity3d.com/Manual/index.html

https://assetstore.unity.com/packages/3d/characters/uma-2-unity-multipurpose-avatar-35611

https://assetstore.unity.com/packages/3d/vegetation/trees/conifers-botd-142076

https://assetstore.unity.com/packages/2d/textures-materials/grass-flowers-pack-free-138810

https://assetstore.unity.com/packages/tools/input-management/first-person-all-in-one-135316

https://assetstore.unity.com/packages/3d/environments/urban/urban-asset-pack-13193?q=urban%20asset&orderBy=1

https://assetstore.unity.com/packages/3d/props/guns/modern-weapons-pack-14233