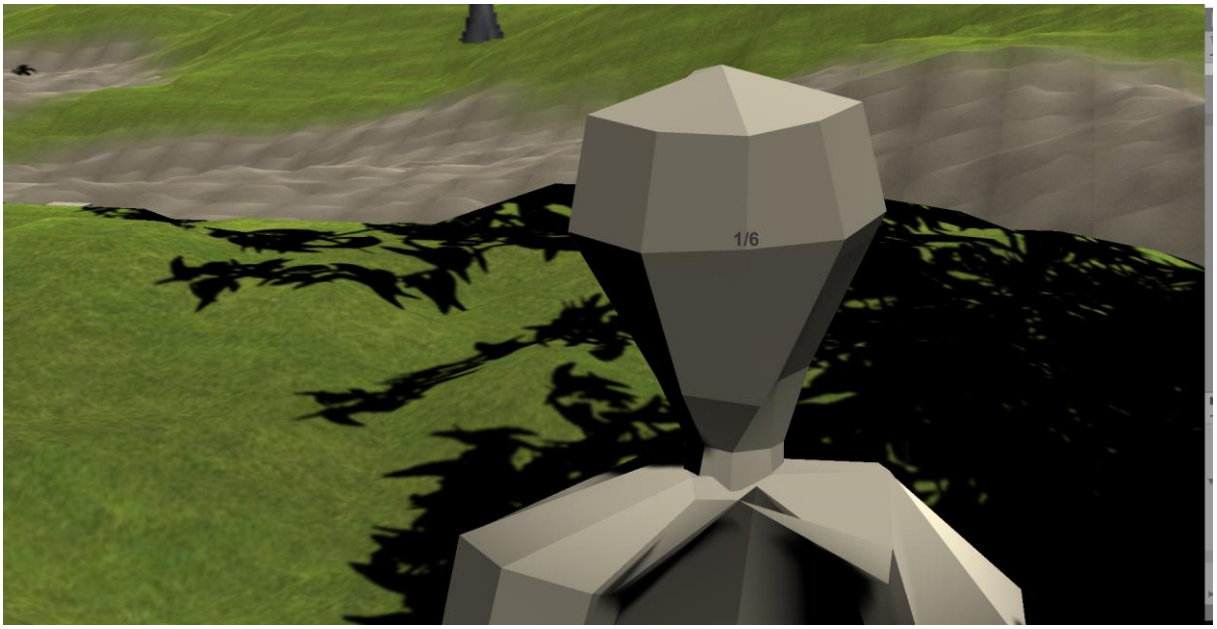
ZTPG- Projekt Teren && Fizyka

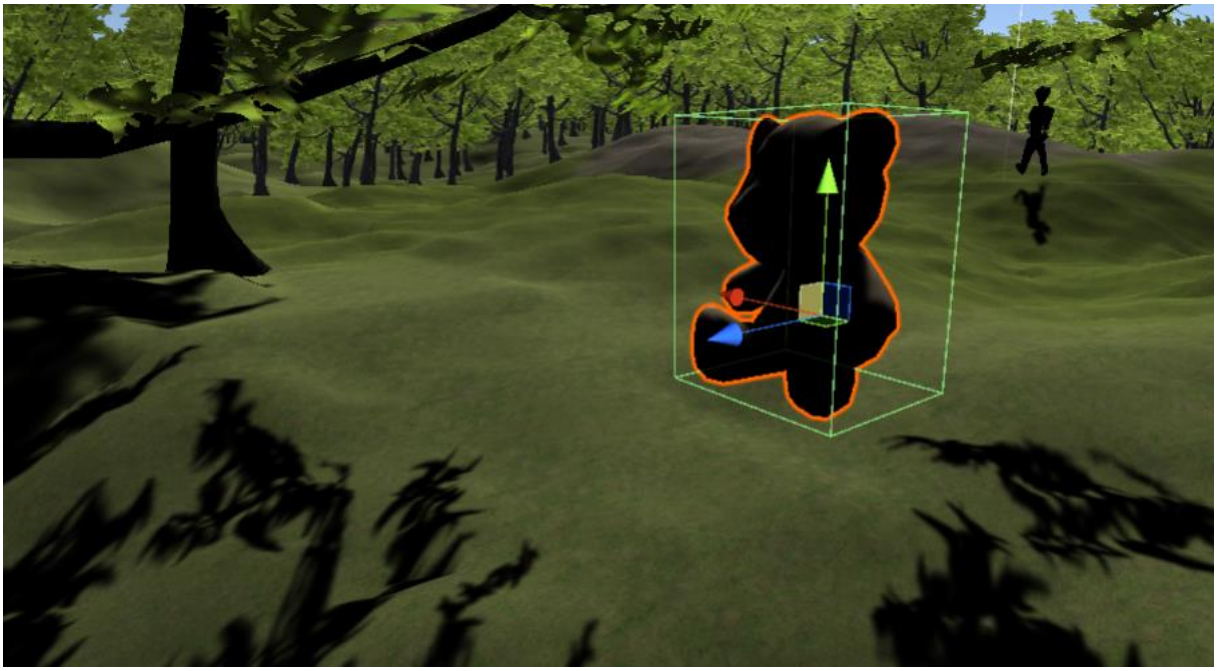
Mateusz Gruchlik

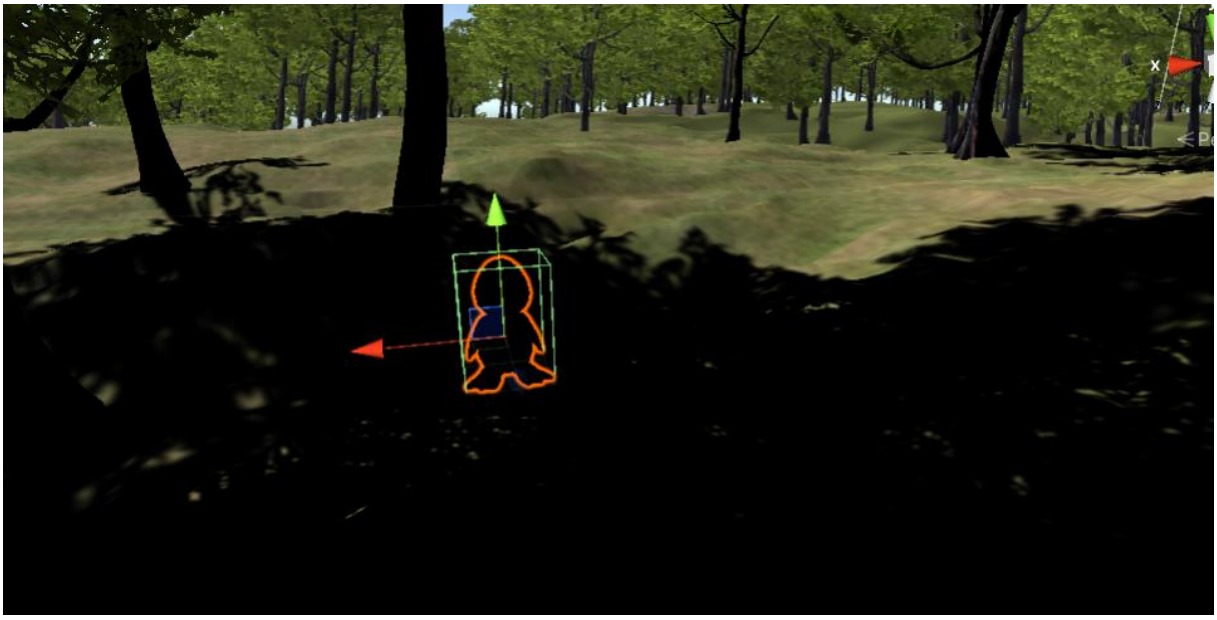
Opis zasad gry

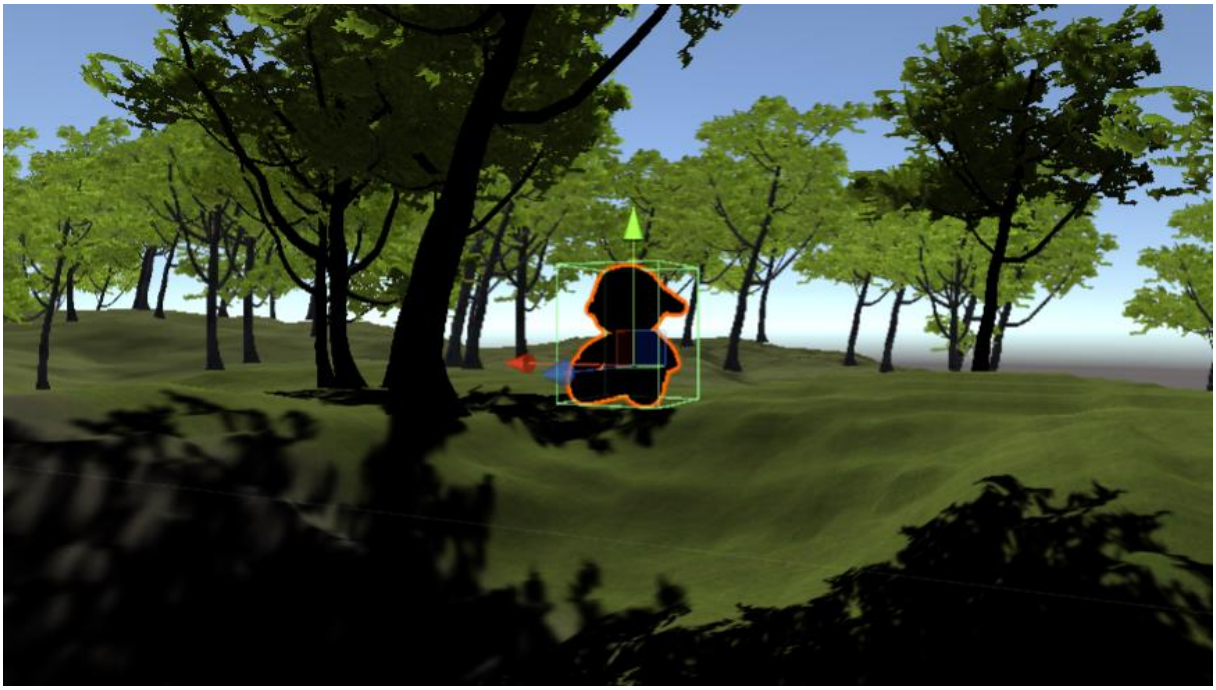
W grze sterujemy pojedynczą postacią humonoidalną.



Naszym celem jest zebranie wyznaczonej liczby obiektów do zebrania (pluszaków) znajdujących się w różnych zakątkach lasu.

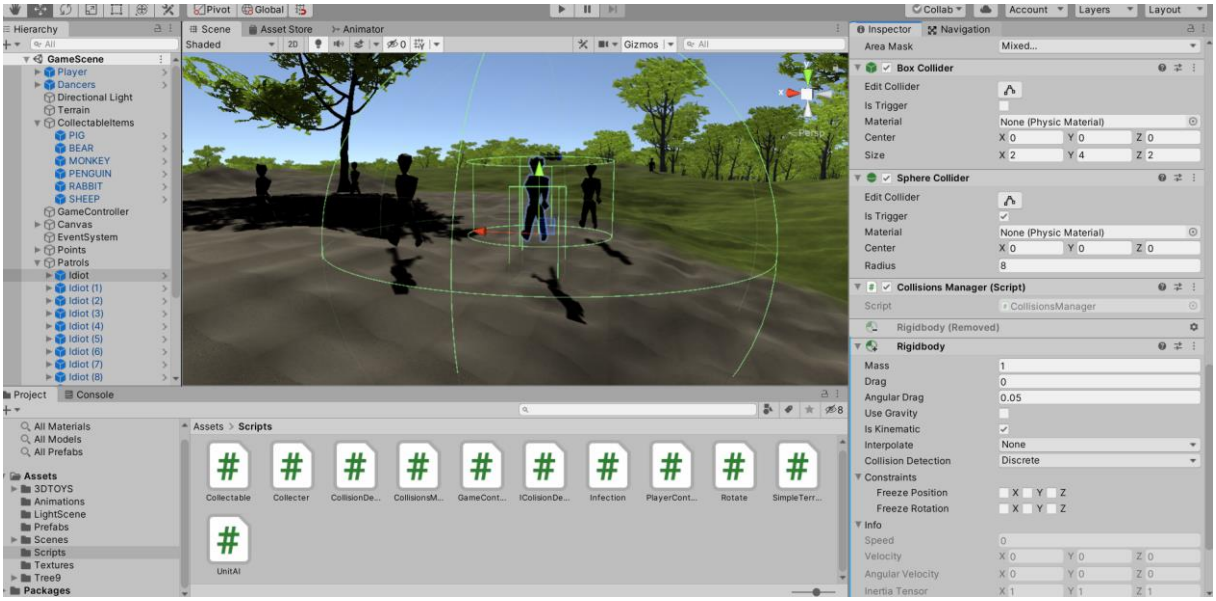




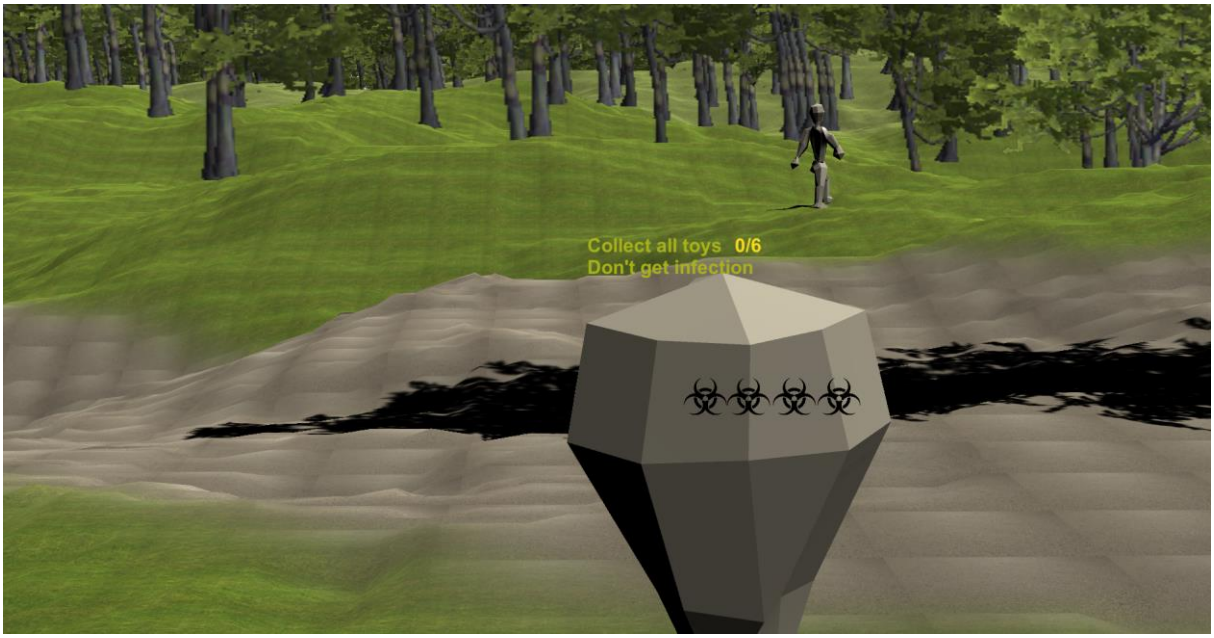


Po zebraniu wszystkich pluszaków gra powiadomi nasz zwycięstwie następującym ekranem.

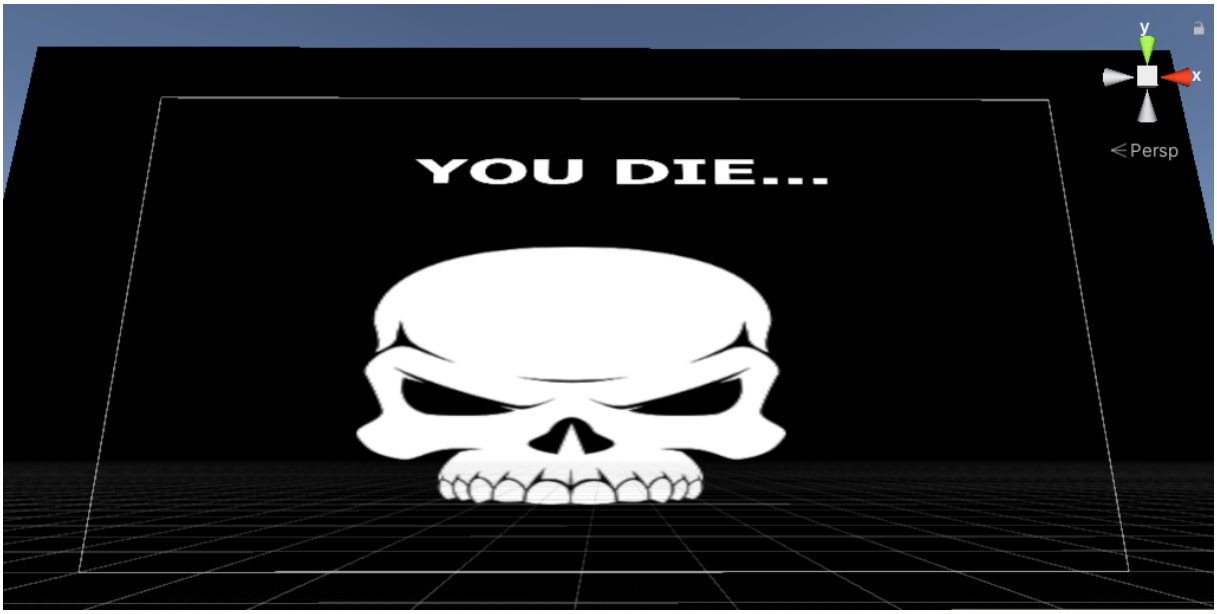
Jednocześnie musimy unikać kontaktu z NPC, którzy potencjalnie mogą przenosić infekcję.



W momencie zetknięcia się z zarażoną osobą (NPC z skryptem Infection), zostaniemy zarażeni (postać gracza otrzyma skrypt Infection).



Pod określonym czasie, gdy poziom infekcji osiągnie wyznaczone maksimum, zostaniemy powiadomieni o przegranej rozgrywce.



Specyfikacja wewnętrzna

Główne klasy

GameController – odpowiedzialny za nadzorowanei przebiegu gry i sprawdzania warunków jego zakończenia

PlayerControler – sterowanie postaci

CollisionManager i implementacje interfejsu **IColisionDecorator** - klasa nadzorująca podstawowe interakcje obiektów świeie gry.

UnitAI - sterowanie przemieszczaniem NPC po świecie gry.

GameController

Funkcja Update

Sprawdza czy zostały spełnione warunki zwycięstwa/przegranej. Dodatkowo uaktualnia informacje na graficznym interfejsie gracza.

```
void Update()
    {
        if (player == null)
        {
            SceneManager.LoadScene(LoseSceneName, LoadSceneMode.Single);
        }
        else if (playerCollecter.CurrentCollected == TotalCollectedItems)
        {
            SceneManager.LoadScene(WinSceneName, LoadSceneMode.Single);
        }

        UpdateText(string.Format(scoreFormat, playerCollecter.CurrentCollected, TotalCollectedItems));

        if(playerInfection == null)
        {
            playerInfection = player.GetComponent<Infection>();
        }
        else
        {
            UpdateInfectionMarkers();
        }
    }
```

Funkcja UpdateInfectionMarkers

Pobiera od obiektu gracza komponent **Infection** (obiekt gracza musi zawierać dany komponent, aby skrypt wszedł do tej metody z funkcji Update). Następnie sprawdza aktualną i maksymalną wartość wskaźnika infekcji i dobiera liczbę markerów na ekranie gracza.

PlayerControler

Update

```
  void Update()
    {
        if (Input.GetKey(KeyCode.W))
        {
            Move(speed, 0);
        }
        else if (Input.GetKey(KeyCode.S))
        {
            Move(-speed, 0);
        }
        if (Input.GetKey(KeyCode.A))
        {
            Move(0, -speed);
        }
        else if (Input.GetKey(KeyCode.D))
        {
            Move(0, speed);
```

```
        }
        else if (Input.GetKey(KeyCode.Space))
        {
            Jump(jumpPower);
        }

        RotateCamera();

        if(transform.position.y <-10)
        {
            transform.position = new Vector3(transform.position.z, 50,
transform.position.z);
        }
    }
```

Funkcje Move, Jump i Rotate Camera

Move zmienia pozycję gracza w świecie gry na osiach X i Z. Co istotne wektor ruchu składający się z parametrów x i z (y jest zero) jest transformowany z układu lokalnego obiektu do układu świata. Dzięki temu niezależnie od ustawienia kamery, postać przy naciśnięciu klawisza W zawsze porusza się w kierunku w którym kamera patrzy. **Jump** nadaje siłę na komponent **Rigibody**. A **RotateCamera** dokonuje obrotu kamerą na podstawie wychylenia myszki.

```
  void Move(float z, float x)
    {
        rigidbody.MovePosition(transform.position + transform.TransformVector(new
Vector3(x, 0, z)));
    }

    void Jump(float force)
    {
        if (rigidbody.velocity.y == 0)
            rigidbody.AddForce(Vector3.up * force);
    }

    void RotateCamera()
    {

        float axisX = Input.GetAxis("Mouse X") * sensitivity;
        float axisY = -Input.GetAxis("Mouse Y") * sensitivity;

        transform.Rotate(0, Time.deltaTime * sensitivity * axisX, 0, Space.World);
        Camera.main.transform.Rotate(Time.deltaTime * sensitivity * axisY, 0, 0,
Space.Self);

        if (Input.GetMouseButtonDown(0))
```

CollisionManager i implementacje interfejsu IColisionDecorator

Przy metodzie Start skrypt uzupełnia swoje listy **collisionsInteractions** i **triggersInteractions** o wszystkie skryptu implementujące interfejs **IColisionDecorator** oraz znajdujące się w tym samym **GameObject**. Przydział do określone listy jest na podstawie flagi **IsTrigger** z interfejsu.
W zależności od tego skypty implementujące interfejs będą wywoływane albo na **OnCollisionEnter** albo na **OnTriggerEnter**.

```
void Start()
    {
```

```
        var allDecorators = GetComponents<IColisionDecorator>();
        collisionsInteractions = allDecorators.Where(x => !x.IsTrigger()).OrderBy(x =>
x.GetPriority()).ToList();
        triggersInteractions = allDecorators.Where(x => x.IsTrigger()).OrderBy(x =>
x.GetPriority()).ToList();
    }

   void OnCollisionEnter(Collision collision)
    {
        foreach (var collisionDecorator in collisionsInteractions)
        {
            collisionDecorator.StartInteraction(gameObject, collision.gameObject);
        }
    }

    void OnTriggerEnter(Collider other)
    {
        foreach (var triggerInteraction in triggersInteractions)
        {
            triggerInteraction.StartInteraction(gameObject, other.gameObject);
        }
    }
```

UnitAI

Metoda ChooseNextPoint

`Określa wybór` następnego punktu docelowego z listy do którego położenia uda się postać NPC.

Sposób wyboru jest zależny od flagi ***IsPatrol***. Gdy *true*, algorytm będzie wybierać następne cele z listy a przy końcu wróci na początek. Przy wartości *false*, punkt docelowy będzie wybrany losowo.

```
    public void ChooseNextPoint()
    {
        if (IsPatrol)
        {
            if (Vector3.Distance(list[currentIndex].transform.position,
transform.position) < 10)
            {
                currentIndex++;
                if (currentIndex >= list.Count)
                {
                    currentIndex = 0;
                }

                agent.isStopped = true;
                agent.destination = list[currentIndex].transform.position;
                agent.isStopped = false;
            }
        }
        else
        {
            if (currentTime >= TimeToChangeDecision)
            {
                currentTime = 0;
                currentIndex = Random.Range(0, list.Count - 1);
```

```
                agent.isStopped = true;
                agent.destination = list[currentIndex].transform.position;
                agent.isStopped = false;
            }
            currentTime += Time.deltaTime;
        }
    }
```

Sceny

GameScene- główna scena gry

WinScene – widok po wygranej rozgrywce

LoseScene – widok po przegranej rozgrywce.

Prefaby

Player – gameobject reprezentujący gracza wraz z gotowymi skryptami i kamerą.

GameControler- pusty obiekt z skrytem GameController. Z podłączonymi elementami UI oraz scenami zwycięstwa/przegrnej.

IDIOT (skrót od „Infection Decorator Inerface Object Test”)- prefab reprezentujący NPC wraz skryptem AI i komponetami fizyki.

Specyfikacja zewnętrzna

Sterowanie postaci odbywa się za pomocą klawiszy **WASD**, którymi przesuwamy postać odpowiednio przód, prawo, tył lewo, myszką, którą obracamy kamerą oraz na przycisk **SPACE** postać wykona skok.

W górnym prawym rogu ekranu będzie wyświetlona informacja o aktualnej liczbie zebranych przedmiotów.

Na początku gry nasza postać nie jest zarażona, ale zmieni się w to momencie bliskiego kontaktu (kilka metrów) z postacią niezależną posiadającą infekcję (NPCz skryptem infection). W dolnym prawym rogu za to będzie wyświetlany stopień naszej infekcji. Gdy na ekranie pojawi się 10 znaczków z symbolem wirusa, postać pozostanie parę sekund przed śmiercią postaci i zakończeniem rozgrywki co grap poinformuje nas następującym ekranem.

Testy

Testowanie NavMeshAgent

Przy podejściu losowania punktów docelowych funkcją random z zakresu obszaru mapy, psotacie niejednokrotnie zatrzymywały się w miejscu. Pomimo różnych parametrów algorytmu (czas czekania, zakres losowania) duża część postaci dalej stała w miejscu. Dopiero ustawienie ręczne różnych punktów na mapie i nawigowanie po nich dało dość zadowalające rezultaty.

Przenikanie przez teren

Postać gracza miała skłonność do spadania przez mapę zwłaszcza przy górzystym terenie. Po ustawieniu w komponencie Rigidbody w obiekcie Player Collison Detection na Continuous Speculative. Ograniczyło to część przypadków, ale wiąż postać wypadała przy najbardziej stromych wzniesieniach. Zmniejszenie prędkości postai z wartości 3 na 1 wyeliminowało 95% przypadków.

Testowanie obszaru kolizji gracza i obiektów do zebrania

Domyślny rozmiar collidera był nieintuicyjny. Gracz musiał dosłownie wejść w środek przedmiotu do zebrania, aby zadziałała kolizja. Dlatego zakres collidera został powiększony dwukrotnie plus zmienio położenie kamery względem postaci.

Testowanie skryptu infekcji i przenikania na inne obiekty

Na początku skrypt nie chciał się wywoływać. Po dodaniu Rigibody dla celu skyptu działał tylko na bardzo bliskim dystansie. Zmiana wywołania na Trigger z Kolizji umożliwiła działanie skryptu na większym obszarze, bez ingerencji w interakcję fizyczną.

Podsumowanie

Najbardziej czasochłonnym elementem projektu okazało się nie teren czy implementacja wymagań z fizyki, ale sam NavMesh oraz próby posłania NPC w wybrane miejsce. Na początku próbowałem przez losowanie dowolnej pozycji w obrębie wymiarów terenów. Następnie sprawdzałem czy wyznaczanie punktu docelowego na podstawie rayhit nie okarze się lepsze. Dopiero umieszczenie ręcznie obiektów 3D robiących za punkty docelowe, pobieranie ich pozycji w skrypcie i przekazywanie do agenta dały zadowalający efekt. Postacie dalej co prawda się zacinają, zwłaszcza na początku. Wynika to, że przy starcie komponenty typu agent liczą jeszcze ścieżki. Dlatego na początku gry można zauważyć, że część postaci stoi.

Projekt można rozwinąć przez dodanie różnych zachowań NC jak agresywny (goni gracza) lub tchórzliwy (ucieka przed innymi), dodanie bonusów wspomagających rozrywkę gracza (wskaźnik na cele gry, spowolnienie tempa infekcji albo podgląd które NPC są zarażone) i przede wszystkim generowanie losowo terenu wraz z rozkładem celów oraz NPC. Obecne funkcjonalności również wymagają poprawy. Sterowanie NPC jest tylko zadowalające. Sterowanie gracza jest strasznie toporne. Można też umieścić więcej interaktywnych elementów związanych z fizyką.