UNIWERSYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI

Jakub Flis
117794

Informatyka

Aplikacja w Unity do generowania syntetycznych, dynamicznych ujęć filmowych do problemów ML

Praca projektowa
Modelowanie i analiza systemów informatycznych

Praca wykonana pod kierunkiem

Mgr inż. Ewy Żesławskiej

Rzeszów 2025

Spis treści

1. Opis założeń projektu

Celem projektu jest stworzenie narzędzia wspomagającego automatyczne generowanie syntetycznych danych wideo w środowisku *Unity*. Projekt zakłada zaprojektowanie systemu, który pozwala na nagrywanie oraz przeglądanie ujęć, przy jednoczesnym zapisie metadanych potrzebnych do trenowania algorytmów widzenia komputerowego. Generowane w nim dane powinny być jak najbardziej różnorodne wizualnie.

Współczesne systemy uczące się, takie jak modele widzenia komputerowego, wymagają ogromnych ilości danych do skutecznego działania. Głównym problemem, jaki projekt stara się rozwiązać, jest czasochłonność i trudność w pozyskaniu dużych zbiorów danych filmowych.

Problem rozwiązywany w ramach projektu związany jest z luką badawczą zarówno w zakresie generacji syntetycznych danych filmowych (obecnie najczęściej wykorzystywany jest symulator *Carla*). Tworzony generator danych pozwoli na ułatwienie badań prowadzonych w przyszłości w tych zagadnieniach. Modele sztucznej inteligencji do analizy ujęć dynamicznych pozwolą na ułatwienie przeprowadzania analiz filmowych oraz dostarczą dodatkowych danych interesariuszom w branży kinematograficznej. System może być użyty przez inżynierów robotyki do ulepszenia analizy danych z sensorów oraz przez uczelnie i badaczy do przygotowania danych do uczenia maszynowego.

Realizacja projektu będzie przebiegać w kilku etapach:

1. Analiza i projektowanie – zidentyfikowanie modułów systemu, przygotowanie diagramów komponentów i procesów (*UML*, *BPMN*).
2. Implementacja modułu planowania ujęć – generowanie trajektorii kamery, wybór punktów centralnych scen, serializacja planów nagrań.
3. Nagrywanie i zapis danych – kontrola kamery, wydajny zapis klatek, eksport metadanych.
4. Realizacja symulacji zachowań przechodniów – *pathfinding*, losowanie animacji z puli.
5. Wprowadzenie testów ujęć – analiza poprawności ujęć w celu przyspieszenia działania aplikacji w związku z odrzucaniem nieprawidłowych scen.
6. Dodanie interfejsu do przeglądu – narzędzie umożliwiające wygodne usunięcie niechcianych ujęć.
7. Końcowe prace nad *UI* i *UX*.

Wynikiem końcowym będzie aplikacja komputerowa stworzona w *Unity*, umożliwiająca użytkownikom generowanie syntetycznych danych filmowych oraz ich adnotacji, które można bezpośrednio wykorzystywać do trenowania i walidacji modeli uczenia maszynowego. Aplikacja będzie generować ujęcia w sposób wydajny oraz w pełni zautomatyzowany, pozwalając na wygenerowanie całego zbioru danych przy użyciu jednego kliknięcia.

Wymagania funkcjonalne

Interfejs użytkownika (edytor *Unity*)

1. Użytkownik może wybierać typy ujęć (*static, arc, push, pan, roll, dolly-zoom, sideways, zoom, vertical*) i środowisk.
2. Użytkownik może zmieniać parametry ujęć dla poszczególnych środowisk.
3. Użytkownik może dostosować ustawienia globalne (np. ścieżka zapisu, *FPS*, rozdzielczość, długość klipów).
4. Użytkownik może wczytywać ustawienia z pliku *JSON*.
5. Użytkownik może planować ujęcia i zapisywać plany do pliku *JSON*.
6. Użytkownik może przeglądać wcześniej wygenerowane klipy i je usuwać.
7. Użytkownik może wizualizować trajektorię kamery oraz środki klastrów postaci w edytorze.
8. Użytkownik może uruchomić proces generowania klipu.

System generowania klipów

1. System odczytuje plany ujęć z pliku *JSON*.
2. System na podstawie planu ujęcia losuje dokładne parametry takie jak pozycja początkowa kamery itp.
3. System generuje klip wideo zgodnie z zaplanowanymi ujęciami.
4. System zapisuje dane o kamerze do pliku *JSON*.
5. System zapisuje klatki wideo.
6. System generuje klastry postaci w scenie.
7. System może automatycznie usuwać klipy, na których postacie są zakryte.
8. System obsługuje *pathfinding* przechodniów i losuje dla nich punkty docelowe.
9. System zarządza *prefabami* przechodniów i jest w stanie dynamicznie ich instancjonować.
10. System losuje i nakłada animacje na przechodniów.
11. System stosuje losowe efekty *post-processing*.

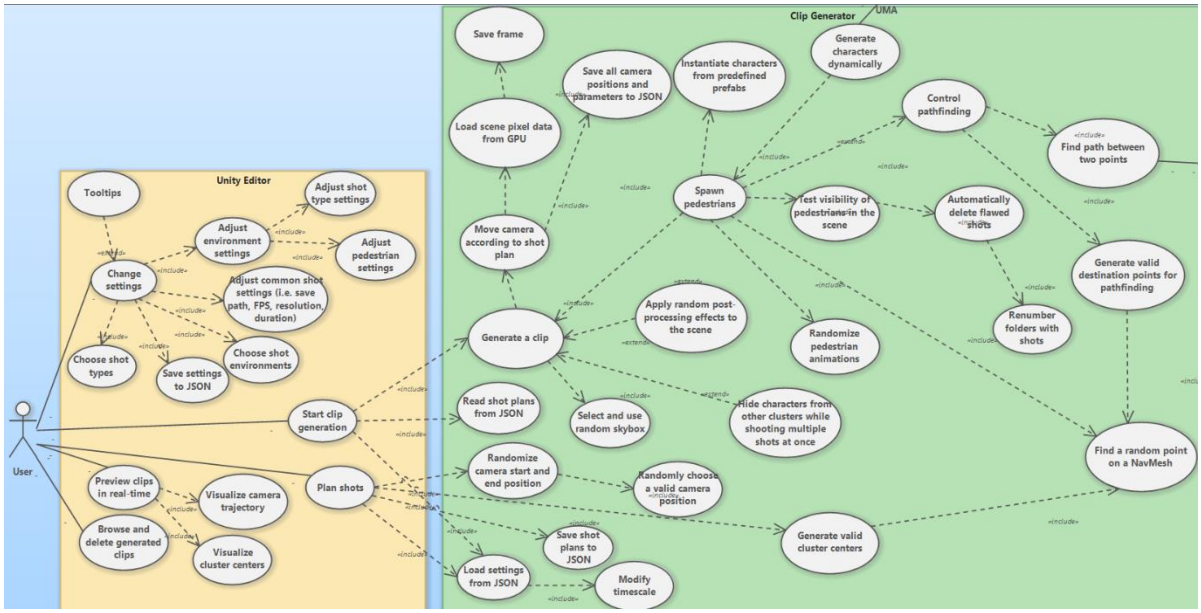
12. System obsługuje usuwanie folderów i ich późniejszą renumerację.
13. System znajduje poprawne, losowe pozycje na siatce *NavMesh* dla poruszających się postaci.
14. System kontroluje *pathfinding* i omijanie przeszkód.
15. System zarządza czasem symulacji (przyspieszenie nagrania).

Wymagania niefunkcjonalne

1. Aplikacja powinna działać płynnie na podstawowym komputerze (*NVIDIA RTX* 1050).
2. Aplikacja powinna obciążać głównie kartę graficzną.
3. Aplikacja powinna umożliwiać eksport co najmniej 30 klatek na sekundę w rozdzielczości *HD*.
4. System powinien być na tyle niezawodny i zautomatyzowany, by był w stanie działać przez całą noc.
5. Dane wyjściowe powinny być w co najmniej 97% przypadków poprawne i nadające się do zadań *ML*.
6. Dane muszą być różnorodne ze względu na postacie, animacje, środowiska i parametry trajektorii kamery.
7. Aplikacja powinna być modularna i pozwalać na łatwe przystosowanie do innych rodzajów zastosowań *ML*.
8. Stosowane *assety* powinny opierać się na licencji *royalty-free*.

2. Modelowanie systemu i jego otoczenia

Rysunek 1. Diagram przypadków użycia



Definicje aktorów

AKTOR	OPIS	PRZYPADKI UŻYCIA
User	Może zmieniać i zapisywać ustawienia systemu, zlecać planowanie i nagrywanie ujęć, a także podglądać postępy i oglądać wyniki końcowe	- PU Change settings <<extend>> z PU Tooltips - PU Start clip generation - PU Plan shots - PU Preview clips in real-time - PU Browse and delete generated clips
UMA (*Unity Multipurpose Avatar*)	System do dynamicznego tworzenia proceduralnie generowanych postaci	- PU Generate characters dynamically
Unity.AI	System obsługujący *pathfinding* oraz analizę terenu	- PU Find path between two points

Scenariusze przypadków użycia:

PU Zmiana ustawień generatora ujęć

OPIS:

CEL – Zmiana ustawień generatora ujęć przy użyciu edytora *Unity*

WS: Poprawne włączenie sceny generatora ujęć

WK: Wygenerowanie pliku *JSON* z ustawieniami użytkownika oraz przygotowanie programu do dalszej pracy

PRZEBIEG:

Użytkownik wybiera obiekt *GlobalSettings* z hierarchii obiektów. W dolnej części komponentu *GlobalVars* zaznacza środowiska, w których nagrane mają zostać ujęcia. Następnie w komponencie *SettingsManager* wybiera typy ujęć na liście *ShotTypes* do nagrania.

Użytkownik zmienia rozdzielczość, prędkość działania programu (ang. *timescale*) oraz włącza efekty *post-processing*. Następnie wybiera w hierarchii obiekt środowiska (mapy) Port. Zawiera on komponent *Environment Settings*. Dla wybranych wcześniej typów ujęć dostosowuje parametry takie jak liczebność ujęć, kąt oraz odległość kamery od punktu zainteresowania. Następnie użytkownik uruchamia aplikację. Plik *JSON* z wygenerowanymi ustawieniami znajduje się w katalogu *root* projektu.

PU Wygenerowanie planów ujęć

OPIS:

CEL – Zapisanie plików *JSON* zawierających dane do nagrania ujęć filmowych

WS: Wybranie typów ujęć, środowisk (map) oraz odpowiednich ustawień

WK: Wygenerowanie plików *JSON* dla każdego wybranego typu ujęć

PRZEBIEG:

Użytkownik wybiera obiekt *Main* z hierarchii obiektów. Następnie odznacza opcję *Make Shots* i zaznacza opcję *Make Plans*. Następnie uruchamia aplikację. Plik *JSON* z planami ujęć znajduje się w katalogu z ujęciami z obecną datą.

PU Wygenerowanie ujęć

OPIS:

CEL – Zapisanie klatek z ujęć w uporządkowanej formie

WS: Wybranie typów ujęć, środowisk (map) oraz odpowiednich ustawień (w tym *Show Camera Preview* oraz *Show Viewer*)

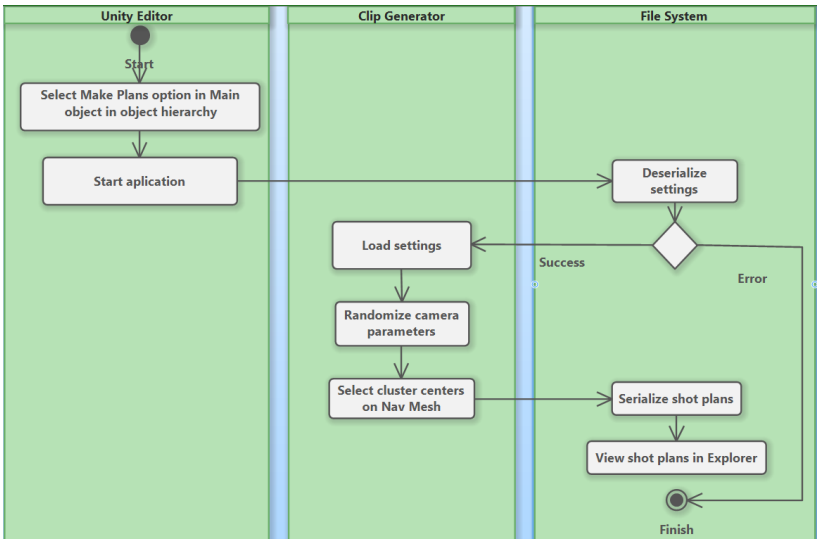
WK: Zapisanie klatek z ujęć oraz pozycji kamery w wybranym katalogu oraz obejrzenie nagranych ujęć w interfejsie edytora

PRZEBIEG:

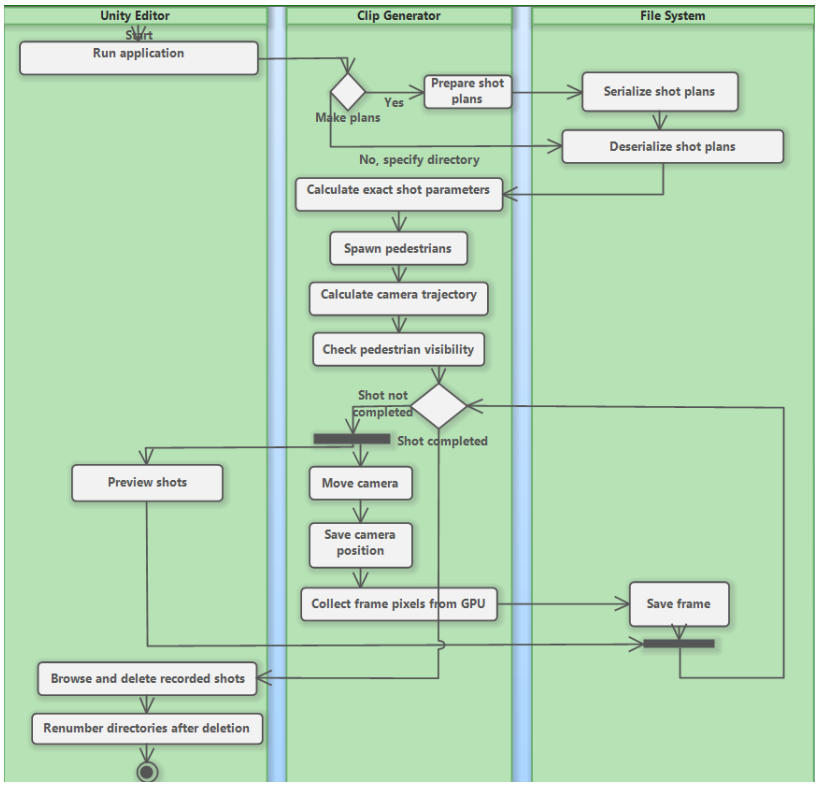
Użytkownik uruchamia aplikację. W czasie rzeczywistym nagrywane są wybrane typy ujęć w wybranych środowiskach. Użytkownik widzi ruchy kamery oraz nagrywany przez nią obraz. Po nagraniu wszystkich ujęć pojawia się interfejs przeglądania ujęć. Użytkownik usuwa ujęcia o niższej jakości. Następnie naciska przycisk „Zakończ i zrenumeruj”. Numery katalogów

zostają poprawione. Ujęcia oraz pozycje kamery zapisane zostały w wybranym katalogu z ujęciami.

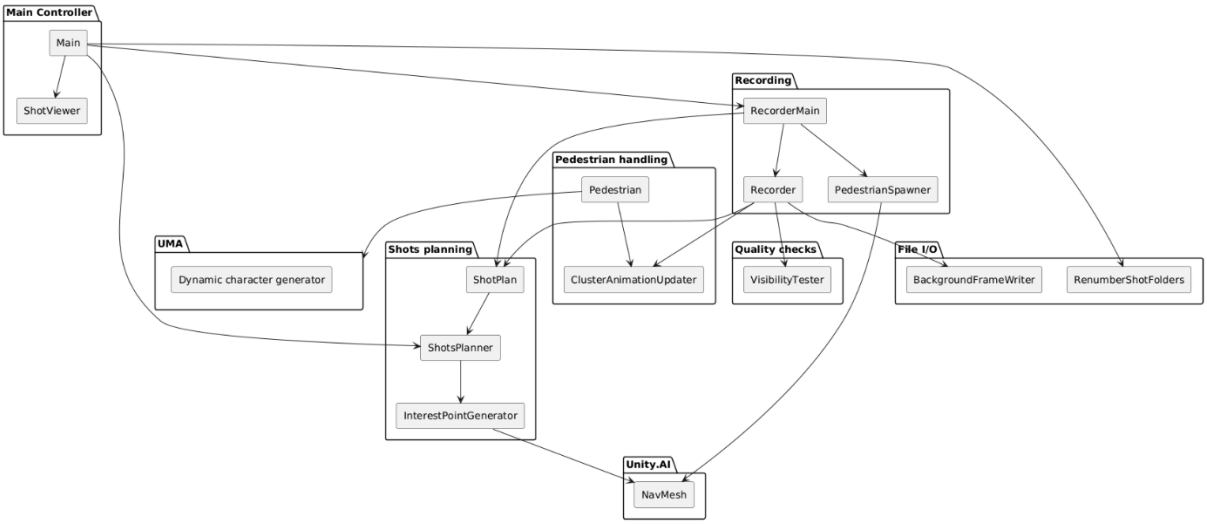
Rysunek 2. Diagram aktywności planowania ujęć



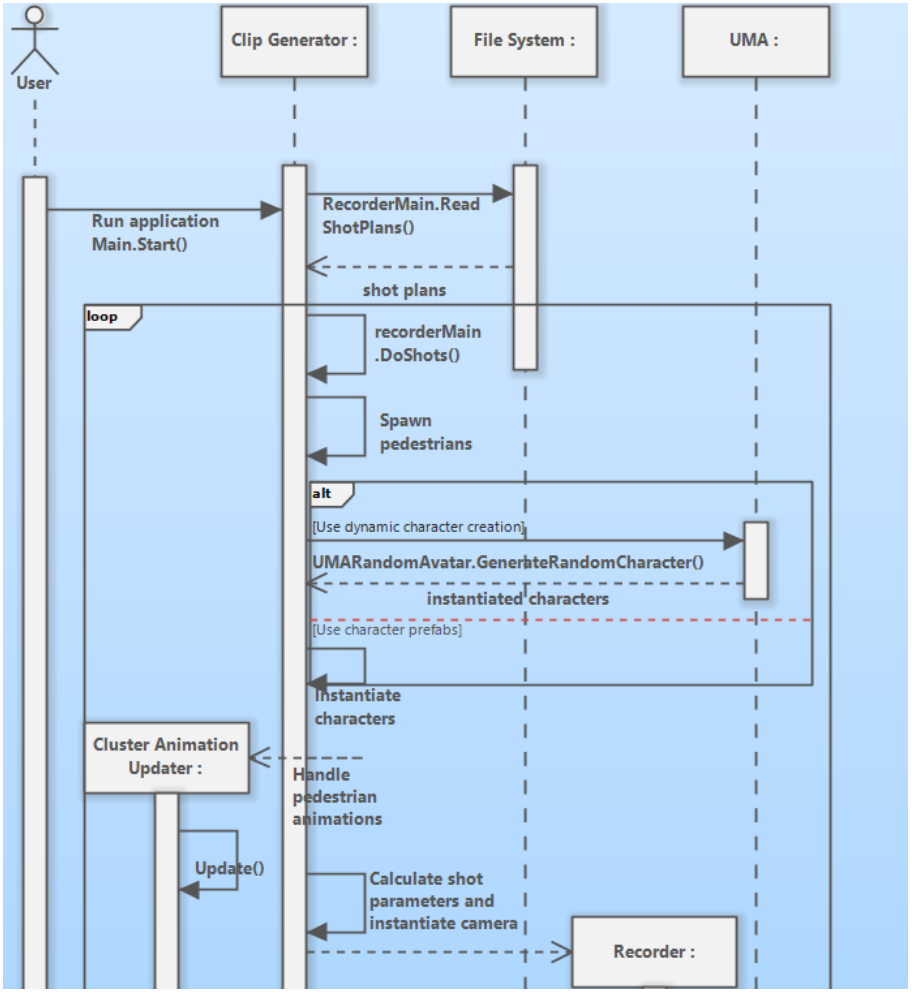
Rysunek 3. Diagram aktywności generowania pojedynczego ujęcia

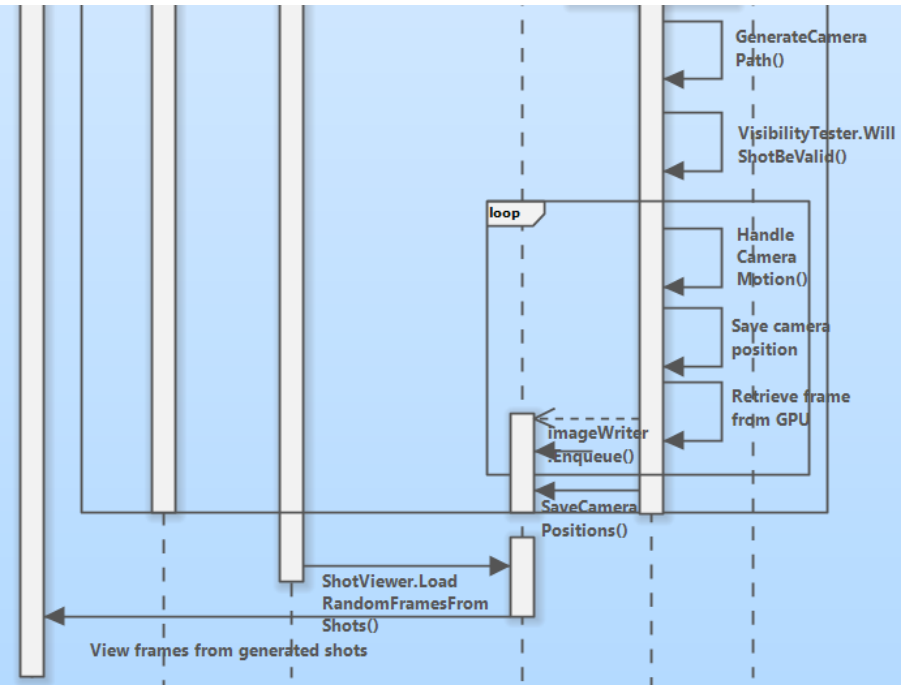


Rysunek 4. Diagram komponentów



Rysunek 5. Diagram sekwencji generowania ujęć





3. Opis struktury projektu

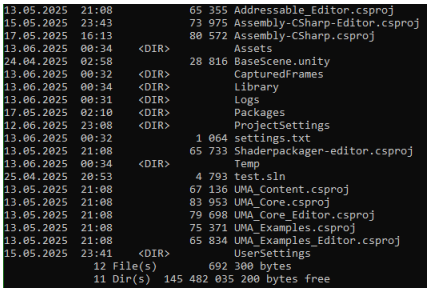
W projekcie wykorzystano następujące narzędzia:

- Silnik i edytor *Unity* 6000.0.32f1,
- Potok renderowania grafiki w *Unity – Built-In Render Pipeline*,
- *Visual Studio Code* – edytor tekstu do pracy nad skryptami *C#* w *Unity*,
- Wtyczka *Unity* for *Visual Studio Code* w wersji 1.12,
- Animacje przechodniów: *Adobe Mixamo* (licencja *royalty-free*), a także *assety* z *Unity Asset Store*: *Various Attitudes animations (Motion Cast - FREE01), FREE - 32 RPG Animations,*
- *Unity Multipurpose Avatar (UMA)* 2.14f4 – elastyczne narzędzie oraz framework do proceduralnego generowania w czasie rzeczywistym realistycznie wyglądających postaci,
- Pakiet *Unity AI Navigation* 2.05 – pakiet zawierający komponenty i skrypty związane z *pathfindingiem* oraz siatką nawigacji (ang. *navmesh*),
- *Newtonsoft Json* 3.2.1 – pakiet ułatwiający serializację oraz deserializację *JSON* złożonych, zagnieżdżonych struktur danych,
- Pakiet *Unity Post Processing* 3.4.0 – pakiet dostarczający implementacje podstawowych efektów *post-processing* oraz związane z nimi komponenty sceny,
- *ColorSkies – asset* z *Unity Asset Store* zawierający 8 *skyboxów*,

- mapy pochodzące z *Unity Asset Store*: *3D Scifi Kit Starter Kit* (w projekcie: *SpaceStation*), *Green Forest* (*Forest*), *LowPoly Mysterious Dungeon* (*Dungeon*), *Mars Landscape 3D* (*Mars*), *Old Sea Port – Environment* (*Port*), *RPG/FPS Game Assets for PC/Mobile (Industrial Set v2.0)* (*Factory*), *Sci-Fi Styled Modular Pack* (*Outpost*),
- *Unity Profiler* – narzędzie do monitorowania wydajności i zużycia zasobów w scenie. Zostało wykorzystane do określenia wąskiego gardła (ang. *bottleneck*) podczas nagrywania i zapisu ujęć. W przypadku tego projektu jest nim wydajność karty graficznej potrzebna podczas przetwarzania i zapisu ogromnej liczby pikseli na dysk,
- Nie wykorzystano żadnego systemu kontroli wersji – projekt jest realizowany w zespole 1-osobowym. Bardzo duży rozmiar plików oraz częste zmiany w strukturze projektu utrudniają tworzenie repozytorium,
- Dane zapisywane są do formatu *.jpg* 95,
- Minimalne wymagania sprzętowe: system operacyjny kompatybilny z *Unity* 6000.0.32f1, 8 GB RAM, karta graficzna GTX 1050 lub lepsza, procesor co najmniej dwurdzeniowy, 7 GB miejsca na dysku.

Struktura projektu

Rysunek 6. Struktura katalogu root



Katalog *root* w większości opiera się na strukturze projektu typowej dla projektów *Unity*. Zawiera on:

- *Assets* – najważniejszy katalog projektu, zawierający wszystkie skrypty, *assety*, kody pakietów,
- *Library* – folder zawierający kody używanych bibliotek,
- *Packages* – folder zawierający *manifest.json* służący do śledzenia wykorzystywanych bibliotek,
- *ProjectSettings* – katalog śledzący ustawienia silnika *Unity* i dotyczące całego projektu (np. ustawienia grafiki),

- *UserSettings* – miejsce przechowywania ostatnich ustawień edytora *Unity* (np. układ okien, otwarte okna, ustawienia sceny zmieniane przy pomocy edytora),

Ponadto w katalogu utworzono:

- *CapturedFrames* (nazwa zmieniana w ustawieniach sceny) – katalog zapisu generowanych ujęć,
- *settings.txt* (nazwa zmieniana w ustawieniach sceny) – ustawienia generatora ujęć, wczytywane przed rozpoczęciem generowania klipów.

Rysunek 7. Struktura katalogu Scripts

Najważniejsze utworzone skrypty (katalog */Assets/Scripts*):

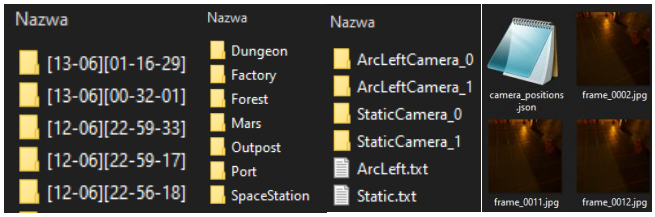
- *BackgroundFrameWriter* – klasa służąca do zapisu klatki asynchronicznie,
- *ClusterAnimationUpdater* – skrypt ustawiający szybkość animacji ruchu dla biegających przechodniów, a także cyklicznie (po zakończeniu 1 cyklu animacji) losujący animację z puli dla stojących przechodniów,
- *InterestPointGenerator* – skrypt zawierający funkcję *GenerateInterestPoints()* służącą do losowania centralnych punktów każdej sceny filmowej, a także pomniejsze funkcje służące do generowania poprawnych punktów leżących na siatce nawigacyjnej (ang. *NavMesh*),
- *Main* – skrypt główny, kontrolujący całą aplikację od początku do końca jej działania, zawiera w szczególności funkcję generyczną odpowiedzialną za nagranie ujęć podanego typu w podanym środowisku (mapie) *HandleShotType()*,
- *PedestrianSpawner* – zawiera funkcje *SpawnIdlePedestrians()* oraz *SpawnRunningPedestrians()* służące do wygenerowania przechodniów w losowych pozycjach na siatce nawigacyjnej (ang. *NavMesh*) w promieniu *ClusterRadius* wokół centralnego punktu sceny/ujęcia (w projekcie nazywany środkiem klastra lub punktem zainteresowania), a także przypina do biegających przechodniów funkcję *CheckDestinationReached()*, która za pośrednictwem funkcji *Invoke()* służy do

kontrolowania *pathfindingu* i znajdowania nowych punktów docelowych dla biegających przechodniów,

- *PostProcessingRandomizer* – skrypt zawierający funkcję *ApplyRandomPostProcessingEffects()*, która nadaje scenie dwa losowo wybrane efekty *post-processing* o losowej sile,
- *Recorder* – skrypt, który asynchronicznie steruje kamerą w ujęciu oraz zapisuje klip. Działaniem skryptu steruje funkcja *Record()*, która: generuje trajektorię kamery funkcją *GenerateCameraPath()*, zleca test poprawności ujęcia funkcji *VisibilityTester.WillShotBeValid()* i usuwa niepoprawne, a także wywołuje funkcję asynchroniczną *CaptureFramesAsync()*, która działa przez określoną liczbę sekund (zmienna *duration*) aż do zakończenia ujęcia. W ramach tej funkcji asynchronicznej wywoływana jest funkcja *CaptureFrame()*, która przesuwa kamerę do aktualnej pozycji funkcją *HandleCameraMotion()*, zapisuje obecną pozycję kamery do pliku *JSON* funkcją *SaveCameraPosition()*, a następnie pobiera asynchronicznie aktualną klatkę z karty graficznej przy użyciu *AsyncGPUReadback.Request()*. Funkcja *HandleCameraMotion()* wizualizuje trajektorię kamery oraz punkt centralny ujęcia z użyciem *Debug.DrawRay()*,
- *RecorderMain* – skrypt przygotowujący kamerę oraz ujęcie na podstawie planu ujęcia odczytanego za pomocą funkcji generycznej *ReadShotPlans()*. Rolę sterującą pełni funkcja generyczna *DoShots()*. Losowany jest *skybox* z puli *GlobalVars.skyboxList*, nakładane są efekty *post-processing* ze skryptu *PostProcessingRandomizer*, generowani są przechodnie skryptem *PedestrianSpawner*, tworzona jest kamera wraz ze wszystkimi potrzebnymi komponentami, zlecane jest aktualizowanie animacji przechodniów kamerze skryptem *ClusterAnimationUpdater*, śledzone są wszystkie niepoprawnie nagrane ujęcia (testowane przez *VisibilityTester* w *Recorderze*) w zmiennej *failedIds*, a także na podstawie szczątkowych parametrów z planu ujęcia obliczane są dokładniejsze parametry trajektorii ruchu kamery w funkcji *PrepareSettings()*,
- *RenumberShotFolders* – skrypt, który funkcją *Renumber()* poprawia numerację katalogów z ujęciami podanego typu w przypadku, gdy część katalogów została usunięta. Dla uniknięcia błędów do przeniesienia danych stosowane są katalogi tymczasowe (ang. *temp*),

- *ShotsPlanner* – skrypt, który dla podanego środowiska (mapy) oraz typu ujęcia planuje ujęcia poprzez wylosowanie podstawowych parametrów w funkcji generycznej *PlanShots()*, które pozwalają na późniejsze nagranie ujęcia w sposób deterministyczny. Utworzone plany serializowane są funkcją generyczną *SaveShotPlans()*,
- *ShotViewer* – skrypt wczytujący interfejs do przeglądania świeżo nagranych ujęć – dla każdego ujęcia losowane są 4 klatki funkcją *LoadRandomFramesFromShots()*. Użytkownik ma możliwość ręcznego usunięcia ujęć, które nie odpowiadają jego standardom (np. słaba widoczność obiektów w scenie),
- *VisibilityTester* – skrypt, który w funkcji *WillShotBeValid()* zleca 2 testy widoczności przechodniów w scenie (po wygenerowaniu przechodniów, ale przed rozpoczęciem procesu nagrywania). Na podstawie listy obiektów *CameraPose* opisujących przyszłą trajektorię kamery oraz przy wykorzystaniu fałszywej kopii kamery testowane są: kolizje obiektów z kamerą w funkcji *IsCameraColliding()*, a także występowanie przeszkód na linii widoczności (ang. *line of sight*) między kamerą a każdym z przechodniów w funkcji *ArePedestriansOccluded()* (wykorzystano *raycasting*).

Rysunek 8. Struktura katalogu z ujęciami



Przy każdym uruchomieniu aplikacji tworzony jest katalog z obecną datą według formatu *[MM-dd][HH-mm-ss]*, który łatwo podlega sortowaniu po miesiącu. Wewnątrz tworzone są osobno katalogi dla każdego wybranego środowiska (mapy). W folderze dla danej mapy znajdują się plany ujęć każdego typu oraz bardzo duże liczby katalogów dla każdego nagranego ujęcia niezależnie od typu.Wewnątrz każdego z tych katalogów znajdują się nagrane klatki ujęcia oraz plik *JSON* zawierający informacje o pozycji i parametrach kamery podczas każdej klatki.

Rysunek 9. Struktura plików JSON dla ustawień, planu oraz trajektorii kamery

```
"shotTypes": [
  5
],
"savePath": "CapturedFrames",
"showCameraPreview": true,
"shouldCharactersBeInFrame": false,
"batchSize": 1,
"timeScale": 3.0,
"pathfindingCloseEnoughDistance": 2.0,
"pathfindingRecheckDelay": 0.5,
"minDistanceFromNavMeshEdges": 1.5,
"animMinSpeed": 0.5,
"animMaxSpeed": 0.9,
"resolution": 512,
"fps": 24,
"duration": 1.0,
"antiAliasingStrength": 2,
"minFOV": 40,
"maxFOV": 70,
```

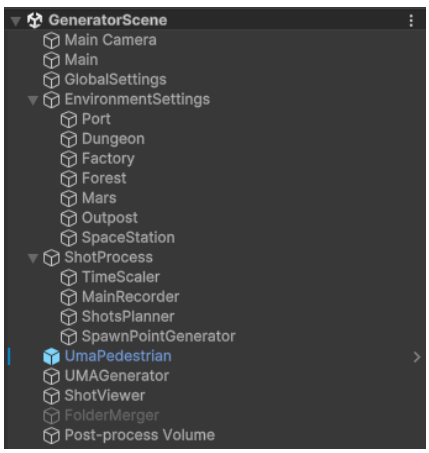
```
{
  "arcDegrees": -30,
  "tiltAngle": 8.0,
  "id": 0,
  "environmentType": 1,
  "fps": 24,
  "duration": 1.0,
  "FOV": 60,
  "shotType": 1,
  "observedPoint": {
    "x": -50.46318,
    "y": 5.091561,
    "z": -4.548716
  },
  "cameraLocation": {
    "x": -51.87802,
    "y": 7.63155365,
    "z": -7.711136
  }
},
```

```
{
  "shotType": 1,
  "shotPlanId": 1,
  "timestamp": 0.0,
  "position": {
    "x": -51.87802,
    "y": 8.046959,
    "z": -7.32686043
  },
  "direction": {
    "x": 0.329351366,
    "y": -0.6879674,
    "z": 0.646705866
  },
  "fov": 60.0
},
```

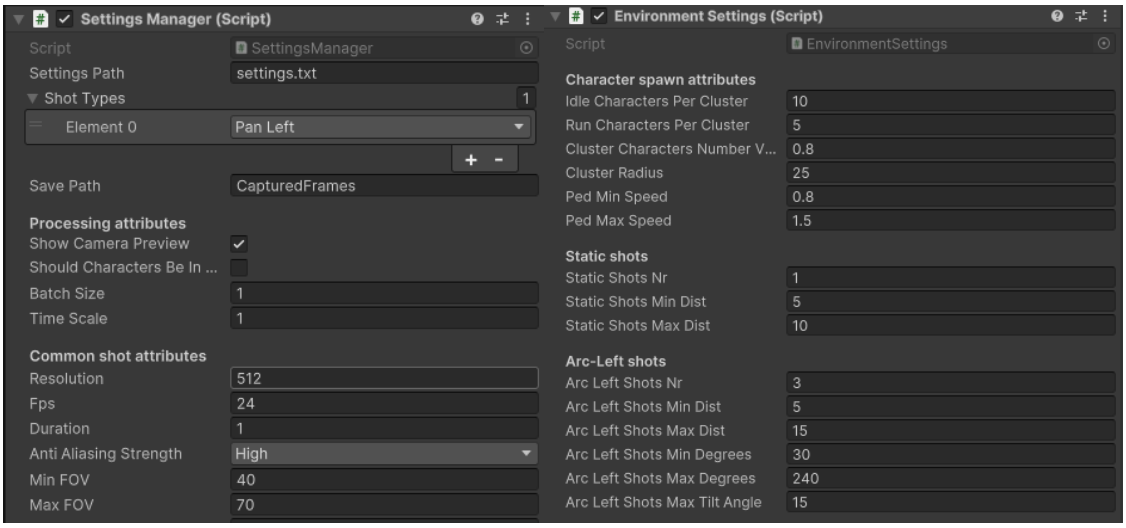
Hierarchia sceny w większości obejmuje obiekty (ang. *gameobjects*), których jedynym komponentem jest pojedynczy skrypt. Podczas pracy z aplikacją ważne są:

- *Main* – pozwala zdecydować przez edytor *Unity*, czy aplikacja ma wygenerować nowe plany ujęć (czy użyć plany z podanej ścieżki), nagrać ujęcia oraz wyświetlić je przy pomocy interfejsu i skryptu *ShotViewer*,
- *GlobalSettings* – zawiera 2 komponenty z ustawieniami możliwymi do edycji przez edytor *Unity*. *GlobalVars* zawiera pulę *skyboxów* oraz animacji, a także pozwala zaznaczyć, w jakich środowiskach mają być przeprowadzone nagrania ujęć. *SettingsManager* zawiera ustawienia związane ze wszystkimi ujęciami niezależnie od środowiska (mapy) – np. rozdzielczość, *FPS*, czas trwania. Opcja *ShowCameraPreview* pozwala wyłączyć podgląd nagrywania dla zwiększenia wydajności aplikacji. Szczególnie ważne jest wybranie typów ujęć *ShotTypes* – zostaną nagrane wszystkie wprowadzone typy ujęć we wszystkich podanych środowiskach (mapach),
- *EnvironmentSettings* – pozwala na sprecyzowanie ustawień ujęć dla każdej mapy. Ustawienia każdego typu ujęć należy podać dla każdego środowiska z osobna, gdyż opcje takie jak odległość kamery są bezpośrednio uzależnione np. od rozmiaru mapy,

Rysunek 10. Hierarchia obiektów sceny



Rysunek 11. Interfejs zmiany ustawień generatora

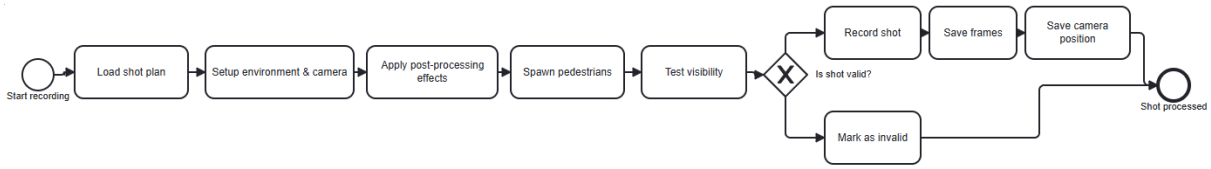


4. Modelowanie procesów biznesowych

Najważniejsze procesy biznesowe zaimplementowane w aplikacji zostały przedstawione przy pomocy notacji BPMN na rysunkach 12 i 13.

Rysunek 12. Diagram BPMN dla planowania ujęć

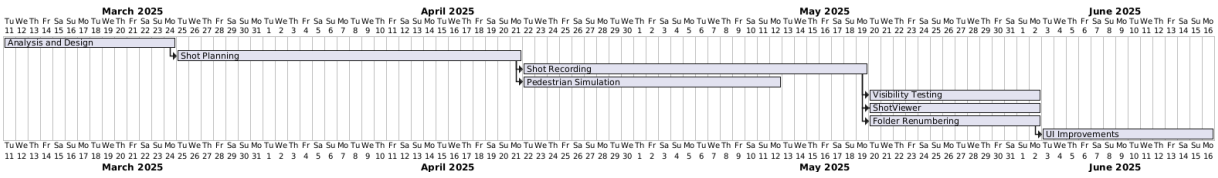
Rysunek 13. Diagram BPMN dla generowania ujęcia



5. Harmonogram realizacji projektu

Harmonogram z podziałem na rodzaje i czas trwania prac zilustrowano przy pomocy diagramu Gantta na rysunku 14.

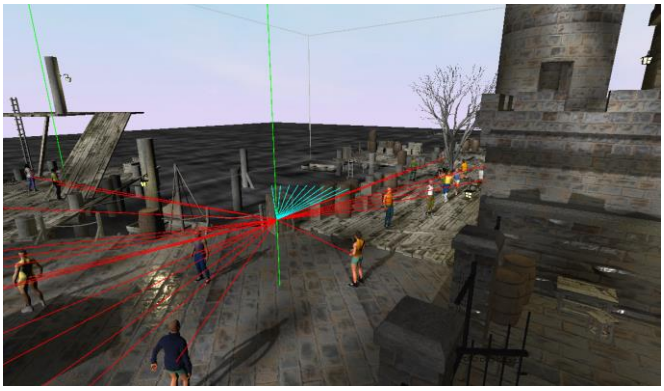
Rysunek 14. Diagram Gannta dla realizacji projektu



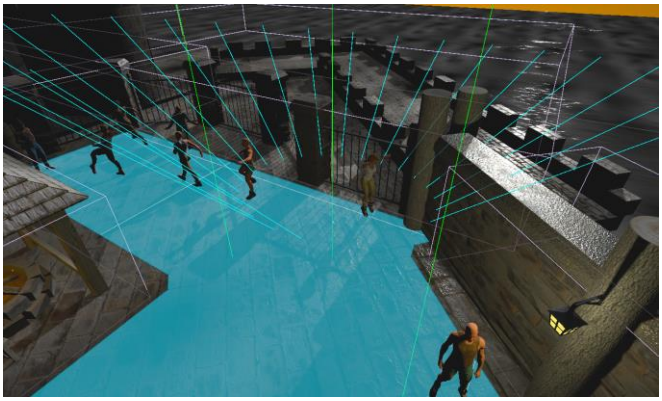
6. Prezentacja warstwy użytkowej projektu

Aplikacja pozwala użytkownikowi na podgląd generowanych ujęć w czasie rzeczywistym. Dla łatwiejszej oceny użytych parametrów możliwy jest podgląd (rys. 15 i 16): trajektorii kamery (kolor turkusowy) oraz środka klastra z postaciami (kolor zielony). Na każdym ujęciu losowany jest 1 z 8 *skyboxów.* Rysunek 16 pokazuje strefy siatki *NavMesh*, po których mogą poruszać się przechodnie (kolor turkusowy). Rysunek 17 demonstruje znaczenie efektów post-processing w różnorodności ujęć. Rysunek 18 przedstawia dostępne środowiska ujęć.

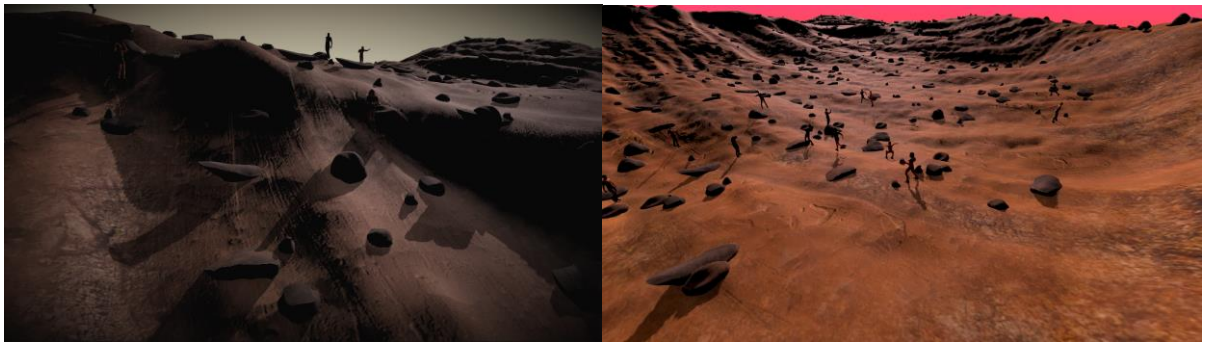
Rysunek 15. Widok podglądu postępu ujęcia typu „pan" w środowisku Port



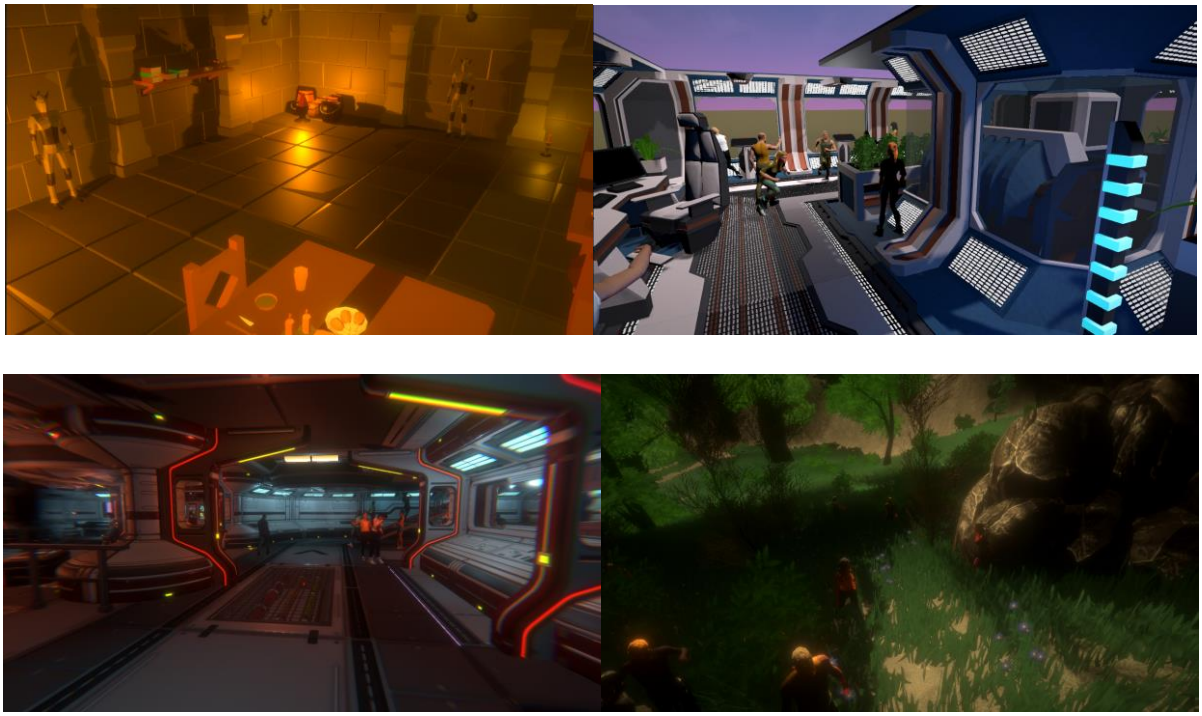
Rysunek 16. Widok podglądu postępu ujęcia typu „arc" w środowisku Port



Rysunek 17. Przykład post-processing w środowisku Mars

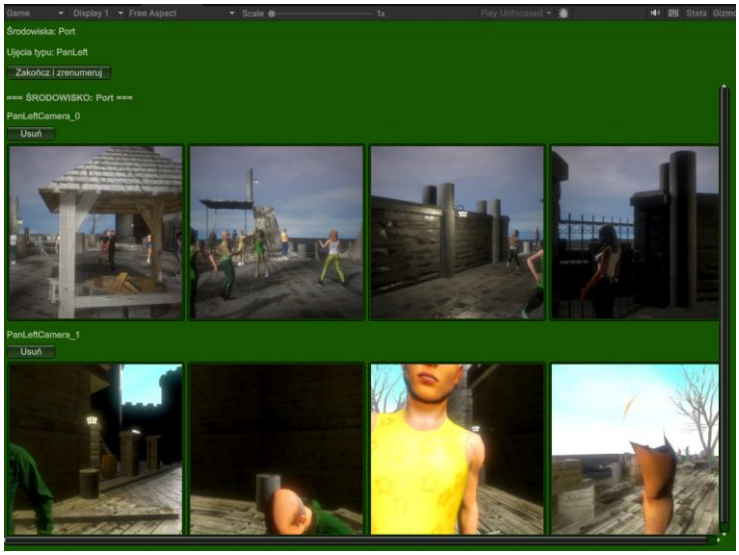


Rysunek 18. Klatki z filmów w środowiskach Dungeon, Outpost, SpaceStation, Forest



Użytkownik po zakończeniu procesu nagrywania ma możliwość przeglądania losowych klatek z nagranych ujęć. Interfejs stwarza możliwość łatwego usunięcia niepożądanych ujęć.

Rysunek 19. Interfejs przeglądania gotowych ujęć



7. Podsumowanie

W ramach projektu opracowano aplikację do generowania syntetycznych danych wideo w środowisku Unity, przeznaczoną do wspomagania uczenia modeli uczenia maszynowego. System opiera się na 2 podstawowych elementach: edytorze użytkownika oraz generatorze ujęć. Udało się w pełni zrealizować wszystkie zaplanowane wymagania funkcjonalne, w tym

możliwość konfiguracji scen, planowania ujęć, dynamicznego generowania postaci z animacjami, zarządzania trajektoriami kamer oraz eksportu adnotacji. Zrealizowano również postawione wymagania niefunkcjonalne — system działa płynnie na budżetowej karcie graficznej, umożliwia generowanie dużych zbiorów danych bez nadzoru oraz zapewnia bardzo wysoką różnorodność i jakość danych wyjściowych. Dzięki modularnej strukturze projekt może być łatwo dostosowany do nowych scenariuszy badawczych.

W ramach dalszych prac nad aplikacją zaleca się stworzenie wersji *build* aplikacji, w której funkcje edytora (zmiana ustawień) zostają przejęte przez system plików konfiguracyjnych JSON. Wygenerowane dane warto uzupełnić o dodatkowe metadane takie jak mapa głębi. Do łatwiejszego dobierania parametrów może przysłużyć się utworzenie systemu *logów* informującego o procencie udanych ujęć. Konieczne jest także dodanie możliwości zapisu w innych formatach poza *jpg*.

Spis ilustracji