



Imię i nazwisko studenta: Przemysław Reszka

Nr albumu: 172038

Poziom kształcenia: Studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Informatyka

Profil: Algorytmy i modelowanie systemów

Imię i nazwisko studenta: Mikołaj Trylewicz

Nr albumu: 171582

Poziom kształcenia: Studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Informatyka

Specjalność/profil: -

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: Wirtualny pokój zagadek (escape room)

Tytuł projektu w języku angielskim: Virtual escape room

Opiekun pracy: dr inż. Jacek Lebiedź

STRESZCZENIE

Celem projektu jest opracowanie dla jaskiń rzeczywistości wirtualnej znajdujących się w Laboratorium Zanurzonej Wizualizacji Przestrzennej aplikacji symulującej różne pokoje zagadek (ang. escape room). Aplikacja powinna pozwalać na konfigurowanie pokoju zagadek przez administratora oraz później umożliwiać udział w zabawie kilku osobom. Uczestnicy zabawy powinni korzystać ze standardowych interfejsów dostępnych w jaskini rzeczywistości wirtualnej, chociażby z kontrolera trzymanego w dłoni (ang. flystick).

Do wytworzenia aplikacji posługiwano się następującymi programami: Blender 2.92, Unity 2018.1.9, Gimp 2.10 oraz Visual Studio 2019.

Wynikiem pracy jest aplikacja przedstawiająca pokój zagadek o tematyce chemicznej pozwalająca na interakcję z przedmiotami oraz działającą w Laboratorium Zanurzonej Wizualizacji Przestrzennej.

Zadaniem gracza jest odblokowanie wszystkich pięciu kłódek znajdujących się na drzwiach pokoju oraz otworzenie sejfu w wyznaczonym czasie.

Przemysław Reszka - udział w rozdziałach 1, 3, 4, 6, 7 oraz indywidualnie podrozdziały 5.1, 5.2.1, 5.2.3, 5.2.4, 5.2.5.

Mikołaj Trylewicz - udział w rozdziałach 1, 3, 4, 6, 7 oraz indywidualnie rozdział 2 oraz podrozdziały 5.2.2, 5.2.6, 5.2.7, 5.3.

Słowa kluczowe: Escape room, pokój zagadek, Laboratorium Zanurzonej Wizualizacji Przestrzennej, aplikacja, Unity, Blender, CAVE, wirtualna rzeczywistość, rozrywka.

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Dziedzina nauk inżynierijno-technicznych, informatyka techniczna i telekomunikacja.

ABSTRACT

The purpose of the project is to develop application that simulates some escape rooms for virtual reality caves that are located in the Immersive 3D Visualization Lab. Application should allow admin to configure room and then enable to take part in game for a few people. Game participants should be able to use standard interfaces available in virtual reality cave such as flystick.

Programs used to develop application: Blender 2.92, Unity2018.1.9, Gimp 2.10 and Visual Studio 2019.

The result of work is application presenting chemistry escape room allowing for interaction with objects working in the Immersive 3D Visualization Lab.

Goal to achieve is to unlock all five padlocks and to open a safe in indicated time.

Przemysław Reszka - participation in chapters 1, 3, 4, 6, 7 and individually in subsections 5.1, 5.2.1, 5.2.3, 5.2.4, 5.2.5.

Mikołaj Trylewicz - participation in chapters 1, 3, 4, 6, 7 and individually in chapter 2 and subsections 5.2.2, 5.2.6, 5.2.7, 5.3.

Keywords: Escape room, application, the Immersive 3D Visualization Lab, Unity, Blender, CAVE, virtual reality, recreation.

SPIS TREŚCI

Wykaz ważniejszych oznaczeń i skrótów	6
1. WSTĘP I CEL PRACY (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)	7
1.1. Założenia ogólne.....	7
1.2. Dostępne symulatory o zbliżonej funkcjonalności	8
2. PROJEKTOWANIE FABUŁY GRY (MIKOŁAJ TRYLEWICZ)	9
2.1. Scena	9
2.2. Fabularne użycie przedmiotów	9
2.2.1. Telewizor z podpowiedziami	10
2.2.2. Telefon z wiadomością od znajomego	11
2.2.3. Układ okresowy pierwiastków	11
2.2.4. Zegar cyfrowy	11
2.2.5. Włącznik światła	11
2.2.6. Obrazy z chemikami.....	11
2.2.7. Kartki z napisanymi związkami chemicznymi	12
2.2.8. Kłódki	12
2.2.9. Sejf	12
2.2.10. Antidotum	12
2.2.11. Waga	12
2.2.12. Statyw na probówki	12
2.2.13. Probówki	12
2.2.14. Papierek wskaźnikowy	12
2.2.15. Skala pH	12
3. CHARAKTERYSTYKA UŻYTYCH NARZĘDZI I TECHNOLOGII INFORMATYCZNYCH (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ).....	13
3.1. Unity	13
3.2. Blender	13
3.3. GIMP	13
3.4. Środowiska programistyczne	14
3.4.1. Visual Studio	14
3.4.2. Rider	14
3.5. System kontroli wersji	14
4. SPECYFIKACJA WYMAGAŃ (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)	15
4.1. Diagram klas	15
5. IMPLEMENTACJA	18
5.1. Modelowanie 3D (Przemysław Reszka)	18
5.1.1. Triangulacja	18
5.1.2. Modyfikatory	19
5.1.3. Eksport modeli	21
5.1.4. Utworzzone modele	22

5.2. Komponenty i interakcje	31
5.2.1. Komponenty Unity (Przemysław Reszka)	31
5.2.2. Klasa MonoBehaviour (Mikołaj Trylewicz)	32
5.2.3. Interfejsy (Przemysław Reszka)	33
5.2.4. Interakcje gracza (Przemysław Reszka)	34
5.2.5. Klasy IGrabable (Przemysław Reszka)	36
5.2.6. Klasy IInteractable (Mikołaj Trylewicz)	39
5.2.7. Klasy dziedziczące tylko po MonoBehaviour (Mikołaj Trylewicz)	41
5.3. Tekstury i materiały (Mikołaj Trylewicz)	42
5.3.1. Materiały emisyjne	42
5.3.2. Materiały transparentne	42
5.3.3. Materiały zanikające	43
5.3.4. Materiały zwykłe	43
6. URUCHOMIENIE APLIKACJI W ŚRODOWISKU CAVE (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)	45
6.1. Funkcje LZWPlib	45
6.2. Okno edytora LZWPlib	46
6.3. Zmiany w kodzie	46
6.4. Testowanie aplikacji na miniCAVE	48
6.5. Przykładowe przejście gry w BigCAVE'ie	49
6.6. Poprawność działania aplikacji	50
6.6.1. Poprawne działające mechanizmy	59
6.6.2. Niepożądane działania	59
7. PODSUMOWANIE (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)	60
Wykaz literatury	61
Wykaz rysunków	61
Wykaz tabel	63

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

1. AR – rozszerzona rzeczywistość (*eng. augmented reality*); system łączący świat rzeczywisty z generowanym komputerowo.
2. VR – wirtualna rzeczywistość (*eng. virtual reality*); obraz sztucznej rzeczywistości stworzony przy wykorzystaniu technologii informatycznych.
3. CAVE – pomieszczenie przeznaczone do zanurzenia się w wirtualnej rzeczywistości.
4. PBR – *eng. Physically Based Rendering*; model cieniowania oddający bardziej realistyczne oświetlenie na materiale.
5. Model – cyfrowa reprezentacja rzeczywistego obiektu za pomocą grafiki trójwymiarowej.
6. Tekstura – bitmapowy obraz definiujący wygląd powierzchni modelu.
7. Materiał – obiekt nakładany na model; definiuje renderowanie powierzchni danego modelu.
8. Rendering – przekształcenie informacji zawartych w plikach na obraz w scenie trójwymiarowej.
9. Shader – program komputerowy wykonywany na GPU opisujący najczęściej zachowanie światła padającego na dany obiekt.

1. WSTĘP I CEL PRACY (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)

Pokoje zagadek (*ang. escape room*) to dosyć nowa forma rozrywki, w której może uczestniczyć dowolna liczba osób. Jednak zazwyczaj preferowana liczba graczy waha się od 2 do 4. Rozgrywka ta polega na poszukiwaniu metody wydostania się z pomieszczenia lub kilku pomieszczeń w określonym czasie, rozwiązując przy tym różne zagadki, od logicznych, poprzez zręcznościowe i na wymagających wiedzy w danym temacie kończąc. Zabawa zazwyczaj trwa 60 minut, po tym czasie należy opuścić pokój. Oczywiście gra może potrwać krócej, jeśli odpowiednio szybko gracze rozwiązają wszystkie łamigłówki i główne drzwi się otworzą.

Początki rozrywki zwanej dzisiaj escape roomem sięgają początków tworzenia gier komputerowych. Pewne elementy rozgrywki od zawsze były widoczne w grach komputerowych, takie jak rozwiązywanie łamigłówek czy interakcja z otoczeniem. Za jedną z pierwszych gier, które skupiały się tylko na ucieczce z zamkniętego pokoju można uznać grę z 1988 roku „*Behind Closed Doors*” autorstwa John'a Willson'a. Jest to gra tekstowa, czyli gra, gdzie na ekranie wyświetlany jest tylko tekst i gracz może tylko wpisywać pewne frazy (np. wstań, podejdź do drzwi itp.), które będą skutkowały pojściem fabuły dalej. W późniejszych latach powstawały podobne gry, najpierw pierwsze gry typu *point-and-click* (wskaż i kliknij) z interfejsem graficznym, najczęściej w 2D, potem tzw. gry *flashowe* (gry na przeglądarkę internetową oparte na programie *Macromedia Flash*, później *Adobe Flash*).

Rozrywka przez większość czasu była dostępna jedynie w wirtualnej rzeczywistości, zatem trafiała do mniejszej liczby ludzi niż w czasach obecnych. Za pierwszą większą próbę przeniesienia zabawy do świata rzeczywistego uważa się projekt Takao Kato. W 2007 roku w Japonii powstał pierwszy obiekt tego typu. Początki nie były łatwe, tylko 6 osób ze 150, które miały okazję uczestniczyć w zabawie, poradziło sobie z wyzwaniem. Jednak pomysł okazał się na tyle dobry, że w przeciągu kilku lat szybko zyskał na popularności i rozprzestrzenił się po całym świecie. Od tego wydarzenia gry planszowe, komputerowe lub nawet tekstowe z tym motywem zaczęły zdobywać wielką popularność nie tylko wśród graczy komputerowych, ale także wśród osób nie związanych z komputerami [9].

1.1. Założenia ogólne

Zrealizowany projekt jest aplikacją symulującą zabawę, która odbywa się w pokojach zagadek. Rozrywka ta nie ma żadnych ograniczeń co do przebiegu rozgrywki, zagadek oraz tematyki. Jedynym ograniczeniem jest wyobraźnia twórców. Przedstawiony escape room zabiera gracza do laboratorium chemicznego, co jednak nie oznacza, że zagadki są jedynie o tej tematyce, więc wiedza z dziedziny chemii nie jest potrzebna do rozpoczęcia zabawy. Najważniejszy jest spryt i umiejętność logicznego myślenia.

Aplikacja docelowo była tworzona tak, aby można było ją uruchomić w Laboratorium Zanurzonej Wizualizacji Przestrzennej, dokładniej w urządzeniu nazywanym BIG CAVE. Dostępne jest tam sterowanie własnym ciałem, po założeniu specjalnych okularów, które są śledzone przez kamery znajdujące się w narożnikach obszaru symulacji, oraz interakcja przy użyciu *flysticka*. Zanim jednak doszło do implementacji potrzebnych opcji do zabawy w LZWP, gra była tworzona na komputerze osobistym. Nie sprawiło to żadnej trudności w implementacji, ponieważ jedyną różnicą rozgrywki na komputerze oraz w CAVE'ie jest sterowanie.

Realizacja projektu została podzielona na 3 główne części:

1. Projektowanie rozgrywki - propozycje narzędzi potrzebnych do realizacji aplikacji, opracowanie fabuły pokoju, wyglądu, dostępnych zagadek oraz ich rozwiązania.
2. Implementacja na komputerze osobistym - tworzenie modeli przedmiotów oraz ich wyglądu, pisanie skryptów odpowiedzialnych za fizykę działającą w grze, interakcję z przedmiotami oraz złączenie wszystkiego w całość.
3. Przeniesienie działającej aplikacji do jaskini wirtualnej rzeczywistości - odpowiednie przerobienie skryptów za pomocą biblioteki LZWPlib oraz dostosowanie rozmiarów przedmiotów.

1.2. Dostępne symulatory o zbliżonej funkcjonalności

Gry o tematyce escape roomów cieszą się dużą popularnością, jednak jest to zazwyczaj zabawa na jeden raz. Przechodzenie kilka razy tego samego poziomu nie daje tyle doznań co za pierwszym razem. Mimo tego rynek gier komputerowych oferuje dużą ilość produktów o podobnej charakterystyce. Znaczną jej ilość można znaleźć na platformie Steam, gdzie gry są publikowane przez profesjonalne studia, jak i niezależnych developerów. Większość gier oferuje zabawę na komputerze osobistym przy użyciu klasycznego sterowania za pomocą klawiatury i myszki.

Dobrym przykładem jest niedawno wydana gra *Escape Simulator* od studia Pine Studio, gdzie gracz sam, lub z przyjaciółmi, może rozwiązywać zagadki w wielu dostępnych pokojach oraz sam je tworzyć. Gra zebrała bardzo dużo pozytywnych opinii, między innymi dzięki przyjemnej oprawie graficznej, łatwej obsłudze oraz dużej liczbie map. Gry tego typu cieszą się większą popularnością na podstawowe platformy, takie jak komputer osobisty czy konsole, ponieważ nie trzeba do nich dokupywać specjalnego sprzętu, takiego jak okulary VR, z którym wiążą się dodatkowe koszty rzędu kilku tysięcy złotych.

Mimo małego zainteresowania na gry przy użyciu specjalnego sprzętu do zabawy w wirtualnej rzeczywistości, nie brakuje ofert tego typu gier na rynku. Nie są one jednak tak bardzo rozbudowane i wspierane przez społeczność, lecz również znajdują swoje grono odbiorców. Jedną z najlepiej ocenianych gier tego typu jest *The Room VR: A Dark Matter* studia Fireproof Games. Oferuje ona rozgrywkę dla jednego gracza tylko przy użyciu sprzętu VR. Aby sterować postacią nie jest wymagane poruszanie się po rzeczywistym pokoju, można siedzieć lub stać. Interakcje z otoczeniem odbywają się poprzez specjalny kontroler imitujący ruch.

Zabawa w Laboratorium Zanurzonej Wizualizacji Przestrzennej oferuje największe zanurzenie w wirtualnym świecie, między innymi dzięki poruszaniu się w specjalnie przystosowanej do tego jaskini. Gier tego typu nie można znaleźć w żadnym sklepie, co czyni je wyjątkowymi.

2. PROJEKTOWANIE FABUŁY GRY (MIKOŁAJ TRYLEWICZ)

Akcja wirtualnego pokoju zagadek odgrywa się w pokoju przypominającym laboratorium chemiczne. Gracz budzi się w laboratorium, przed sobą widzi tylko drzwi, które są zablokowane przez 5 kłódek. Nad drzwiami wiszy zegar, który odlicza czas, dokładnie 60 minut od momentu, w którym się przebudził. Gracz nie czuje się dobrze, podejrzewa, że został zatruty. Uświadamia sobie, iż zostało mu tylko 60 minut życia. W tym czasie musi odnaleźć antidotum i wydostać się z pomieszczenia.

Aby wydostać się z pokoju należy odblokować sejf oraz 5 kłódek znajdujących się przy drzwiach. Do zrealizowania celu gracz może posłużyć się wszystkimi elementami otoczenia. Z większością przedmiotów można wchodzić w interakcje, niektóre można podnosić, trzymać, kłaść, inne otwierać, wpisywać kod lub dzięki interakcji zmieniają wyświetlaną treść.

2.1. Scena

Scena składa się z jednego pokoju. Znajdują się w nim wszystkie przedmioty potrzebne do ukończenia gry. Z punktu widzenia celu gry, rozmieszczenie ich jest dowolne, jednak drzwi, plakaty, obrazy, tablica korkowa, telewizor oraz zegar powinny znajdować się na ścianach oraz lampy powinny znajdować się na suficie. Również stoły powinny być przy ścianie, a na jednym z nich powinien znajdować się sejf, waga lub statyw na próbówkę. Reszta przedmiotów, które można podnosić, są porozrzucane w różnych miejscach na scenie. Przykładowy wygląd sceny znajduje się na rys. 2.1, rys. 2.2 oraz rys. 2.3.



Rysunek 2.1: Pokój zagadek - widok na drzwi.

2.2. Fabularne użycie przedmiotów

Pokój zawiera wiele przedmiotów, z którymi gracz może wchodzić w interakcje. Większość z nich znajduje się na scenie, aby zapełnić przestrzeń, zmylić użytkownika lub służyć za wystrój. W tym podręczniku zostały opisane przedmioty, dzięki którym możliwe będzie rozwiązanie zagadek, co za tym idzie, wyjść z escape roomu w wyznaczonym czasie.



Rysunek 2.2: Pokój zagadek - widok na układ okresowy pierwiastków.



Rysunek 2.3: Pokój zagadek - widok z góry.

2.2.1. Telewizor z podpowiedziami

W wirtualnym pokoju zagadek istnieje możliwość uzyskania podpowiedzi do losowej zagadki przy interakcji gracza z ekranem telewizora.

Początkowo ekran wyświetla nazwę gry, czyli „Pokój zagadek – Laboratorium” (eng. Escape room – Laboratory), w języku angielskim na niebiesko-gradientowym tle. Przy interakcji użytkownika na ekranie zostaje wyświetlona losowa podpowiedź, która utrzymuje się przez 30 sekund. Po upływie tego czasu ekran znów wyświetla nazwę gry. W przypadku, gdy wylosowana podpowiedź nie będzie interesowała gracza, może ponownie wejść w interakcję z ekranem. Wówczas zostanie wyświetlona inna losowa podpowiedź, która również będzie wyświetlana na ekranie telewizora przez 30 sekund.

Podpowiedzi są opcjonalne, nie trzeba z nich korzystać, aby odblokować wszystkie kłódki, zatem aby zniechęcić graczy do ich wyświetlania nie są one w żaden sposób powiązane z kłódkami, tj. gdy zostanie odblokowana kłódka dalej można wylosować do niej podpowiedź.

Wszystkie podpowiedzi są napisane w języku angielskim, aby jak najwięcej użytkowników nie miało problemu z ich zrozumieniem. Tekst jest tak sformułowany, że osoba, która nie zna dobrze języka angielskiego mogła się domyślić treści.

Treść podpowiedzi oraz przypisanie do zagadki:

- Zagadka nr 1 – Look carefully at the message you got from your friend and at laboratory decorations (Popatrz uważnie na wiadomość, którą dostałeś od znajomego oraz na ozdoby w laboratorium).
- Zagadka nr 2 – Combine abc song with some chemistry pattern (Połącz piosenkę o alfabetie z jakimiś wzorami chemicznymi).
- Zagadka nr 3 – Do something with beakers and look around the room (Zrób coś z fiolkami i rozejrzyj się po pokoju).
- Zagadka nr 4 – Search for 4-digit number (Poszukaj 4-cyfrowej liczby).
- Zagadka nr 5 – Open the safe first! (Najpierw otwórz sejf).
- Zagadka nr 6 – Guess the chemist! (Odgadnij chemika).

2.2.2. Telefon z wiadomością od znajomego

Aby rozszyfrować kod do pierwszej kłódki należy znaleźć w pokoju telefon i powiązać wiadomości z czatu tekstowego wyświetlane na ekranie z pierwiastkami widocznymi na tablicy Mendelejewa. Pierwsze słowa w każdej wiadomości wysłanej przez znajomego zaczynają się tak samo jak odpowiedni pierwiastek w układzie okresowym. Kodem do kłódki są odpowiednie cyfry w liczbach atomowych danego pierwiastka. Niektóre pierwiastki mają dwucyfrowe liczby atomowe, w tym przypadku należy wybrać drugą cyfrę, o czym informuje zdanie „it shows only second digit of any number” (pol. Pokazuje tylko drugą cyfrę każdej liczby).

Przy powyższych informacjach można bez problemu odszyfrować kod: He (2), Co (27), S (16), W (74). Zatem kod do kłódki to 2764.

2.2.3. Układ okresowy pierwiastków

Pokazane na nim pierwiastki oraz ich liczby atomowe należy powiązać z wiadomością tekstoną wyświetlana na telefonie.

2.2.4. Zegar cyfrowy

Zegar wyświetla czas, jaki pozostał graczowi do końca gry. Początkowo ustawiony jest na 60 minut, co odpowiada czasowi, który zazwyczaj ma się do wykorzystania w klasycznych pokojach zagadek.

Zegar również pokazuje kod do jednej z kłódek, gdy gracz wejdzie w interakcję z jednym z włączników światła. Wtedy zamiast efektu imitującego zgaszenie się jednej z lamp, zostanie wyświetlony kod (9500) na zegarze i będzie on widoczny przez 5 sekund.

2.2.5. Włącznik światła

Aby na zegarze pojawił się kod należy znaleźć odpowiedni włącznik światła (1 z 3 dostępnych na scenie) i wejść z nim w interakcję.

2.2.6. Obrazy z chemikami

Do odblokowania sejfu należy skorzystać ze zdjęć z twarzami chemików, które znajdziemy na ścianie nad biurkiem. Zdjęcia przedstawiają następujących chemików oraz daty otrzymanych przez nich Nagród Nobla: Eduard Bochner (1907), Maria Skłodowska-Curie (1911), Richard Zsigmondy (1925), Irene Joliot-Curie (1935), John Howard Northrop (1946). Poprawnym kodem, który odblokuje sejf jest data otrzymania Nagrody Nobla przez Marię Skłodowską Curie (1911). W tej zagadce nie ma żadnych zależności. Należy działać metodą prób i błędów. Jedyną podpowiedź może być to, że Skłodowska jest najsławniejszym chemikiem z całej piątki oraz pochodziła z Polski.

2.2.7. Kartki z napisanymi związkami chemicznymi

Do odblokowania kolejnej kłódki należy skorzystać z kartek, które są porozrzucane na scenie. Kartek jest 5, każda posiada dwie strony: pustą oraz z napisem przedstawiającym związek chemiczny. Aby odgadnąć kod do kłódki należy zauważyc, że niektóre pierwiastki w związkach posiadają liczbę częsteczek występujących w danym związku większą niż 1. Teraz wystarczy posegregować alfabetycznie te pierwiastki i ukaże się rozwiążanie. Używane związki: kwas chromowy(VI) (H_2CrO_4), wodorek berylu (BeH_2), wodorosiarczek srebra(I) ($AgHS$), fluorek uranu(VI) (UF_6), fumaran żelaza(II) ($C_4H_2FeO_4$). Duplikaty i pojedyncze wystąpienia pierwiastków należy ignorować. Po prawidłowym uporządkowaniu (C_4 , F_6 , H_2 , O_4) pokazuje się następujący kod: 4624.

2.2.8. Kłódki

Kłódki są najważniejszym przedmiotem w grze. Dzięki nim możliwe jest wygranie gry. Jest ich 5 sztuk, każda z 4 cyfrowym kodem. Gdy kłódka jest zablokowana jedyna możliwa interakcja z nią to zmienianie cyfr w korbce. Po wpisaniu prawidłowej kombinacji, kłódka zostaje odblokowana oraz pojawia się możliwość interakcji polegającej na podniesieniu i poruszaniu się z nią.

2.2.9. Sejf

Aby uzyskać dostęp do antidotum, należy wcześniej odgadnąć 4 cyfrowy kod do sejfu. Po wpisaniu odpowiednich cyfr, następuje odpowiedni dźwięk oraz zostają odblokowane drzwiczki sejfu. Po otworzeniu sejfu gracz ma możliwość zobaczenia antidotum i dokonania wyboru co z nim będzie robił.

2.2.10. Antidotum

Antidotum nie jest dostępne od razu po rozpoczęciu rozgrywki. Jest schowane w sejfie, zatem aby mieć do niego dostęp należy najpierw odgadnąć kod. Następnym krokiem po uzyskaniu dostępu do przedmiotu będzie położenie go na wadze. Wyświetlona liczba (0347) będzie kodem do jednej z kłódek.

2.2.11. Waga

Położenie na wadze antidotum skutkuje pokazaniem się na ekranie kodu do jednej z kłódek.

2.2.12. Statyw na probówki

Statyw jest przedmiotem pomocniczym do rozwiązymania jednego z kodów. Należy w nim umieścić odpowiednie probówki.

2.2.13. Probówki

Probówki są puste oraz z kolorowymi cieczami. Probówki wypełnione należy powładać w odpowiednie miejsca w statywie. Dotknięcie probówek z kolorową cieczą papierkiem wskaźnikowym barwi go na inny kolor.

2.2.14. Papierek wskaźnikowy

Na scenie porozrzucanych jest kilkanaście papierków. Należy znaleźć 4 z nich i przyłożyć do odpowiedniej probówki. Dzięki temu papierek zmieni kolor. Aby odczytać kod należy zwrócić uwagę na plakat przedstawiający skalę pH. Po połączeniu danych wystarczy dopasować cyfry do kolorów patyków aby uzyskać kod (7902).

2.2.15. Skala pH

Pokazaną skalę oraz kolory należy powiązać z kolorowymi papierkami wskaźnikowymi.

3. CHARAKTERYSTYKA UŻYTYCH NARZĘDZI I TECHNOLOGII INFORMATYCZ-

NYCH (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)

3.1. Unity

Unity jest to silnik służący do wytwarzania dwuwymiarowych oraz trójwymiarowych gier komputerowych. Posiada też wsparcie dla wytwarzania gier AR i VR. Unity oferuje dość przyjazny interfejs użytkownika oraz sporą bazę podstawowych komponentów ułatwiających implementację zachowań fizycznych takich jak grawitacja, kolizje itd.

Wykorzystany został program w wersji 2018.1.9 przede wszystkim ze względu na kompatybilność z biblioteką LZWPLib do obsługi jaskini wirtualnej rzeczywistości. Licencja programu pozwala na darmowe użycie dla twórców nie przekraczających swoim budżetem 100 tys. dolarów rocznie.

3.2. Blender

Blender jest to darmowe i otwarto-źródłowe oprogramowanie do tworzenia grafiki 3D. Program zapewnia wsparcie dla pełnego procesu tworzenia 3D — modelowania, *rigging'u*, animacji, symulacji, renderowania, kompozycji, przechwytywania ruchu, a nawet tworzenia gier video. Blender jest wieloplatformowy i działa równie dobrze na systemach Windows, Linux oraz Macintosh.

Do wad oprogramowania należy mało przyjazny interfejs użytkownika i stosunkowo wysoki próg wejścia, co skutkuje wydłużeniem czasu nauki do płynnej pracy z programem.

Program ten został wybrany do realizacji projektu ponieważ:

- oferuje wszystkie niezbędne funkcje do wykonania projektu, jest wszechstronny,
- jest darmowy i otwarto-źródłowy, dzięki czemu posiada wsparcie milionów użytkowników oraz wiele poradników,
- autorzy posiadają doświadczenie w pracy z programem.

Do realizacji projektu wykorzystano program w wersji 2.92.

3.3. GIMP

GIMP jest darmowym i otwarto-źródłowym oprogramowaniem do edycji lub tworzenia obrazów, co również sugeruje jego pełna nazwa: *GNU Image Manipulation Program*. Program umożliwia zaawansowane opcje obróbki obrazu, takie jak praca na warstwach, dodawanie przeźroczystości, wycinanie fragmentów obrazu czy nakładanie tekstu. GIMP jest wieloplatformowy, zatem działa na większości systemów operacyjnych, takich jak GNU/Linux, macOS czy Windows.

Interfejs programu jest dosyć przyjazny dla początkującego użytkownika, który potrzebuje korzystać jedynie z podstawowych funkcji programu. Jako, że GIMP jest darmowym oprogramowaniem, posiada on wsparcie wielu użytkowników oraz szeroka bazę poradników.

Do realizacji projektu wykorzystano program w wersji 2.10.24.

3.4. Środowiska programistyczne

Do tworzenia skryptów aplikacji zostały wykorzystane różne środowiska programistyczne w zależności od preferencji każdego z autorów oraz stanowiska pracy np. laboratorium LZWP.

3.4.1. Visual Studio

Visual Studio jest to bardzo popularne środowisko programistyczne firmy Microsoft. Posiada wiele przydatnych funkcji, takich jak dołączenie do aparatu Unity, co pozwala na testowe uruchomienie aplikacji w Unity w trybie z debuggerem w Visual Studio.

Użyto program w wersji 16 (Visual Studio 2019) na licencji Community – darmowej dla niekomercyjnych zastosowań.

3.4.2. Rider

Rider od *Jetbrains* to wieloplatformowe IDE dla tworzenia aplikacji .NET, .NET Core, Xamarin oraz Unity oparte na *IntelliJ* oraz *ReSharper*. Podobnie do Visual Studio również posiada wsparcie dla uruchamiania aplikacji w Unity w trybie z debuggerem.

Do realizacji projektu wykorzystano Ridera 2021.2.2 na darmowej licencji do celów edukacyjnych

3.5. System kontroli wersji

Do pracy nad jednym projektem przez wielu autorów niezbędne było użycie systemu kontroli wersji wraz ze zdalnym repozytorium. W tym celu użyto darmowej platformy Github – hostingowego serwisu przeznaczonego do projektów programistycznych, wykorzystującego system kontroli wersji Git.

Dodatkowo użyto oprogramowanie Github Desktop umożliwiające wygodne i przejrzyste przeglądanie zmian w przypadku, gdy ich liczba sięga kilkudziesięciu plików.

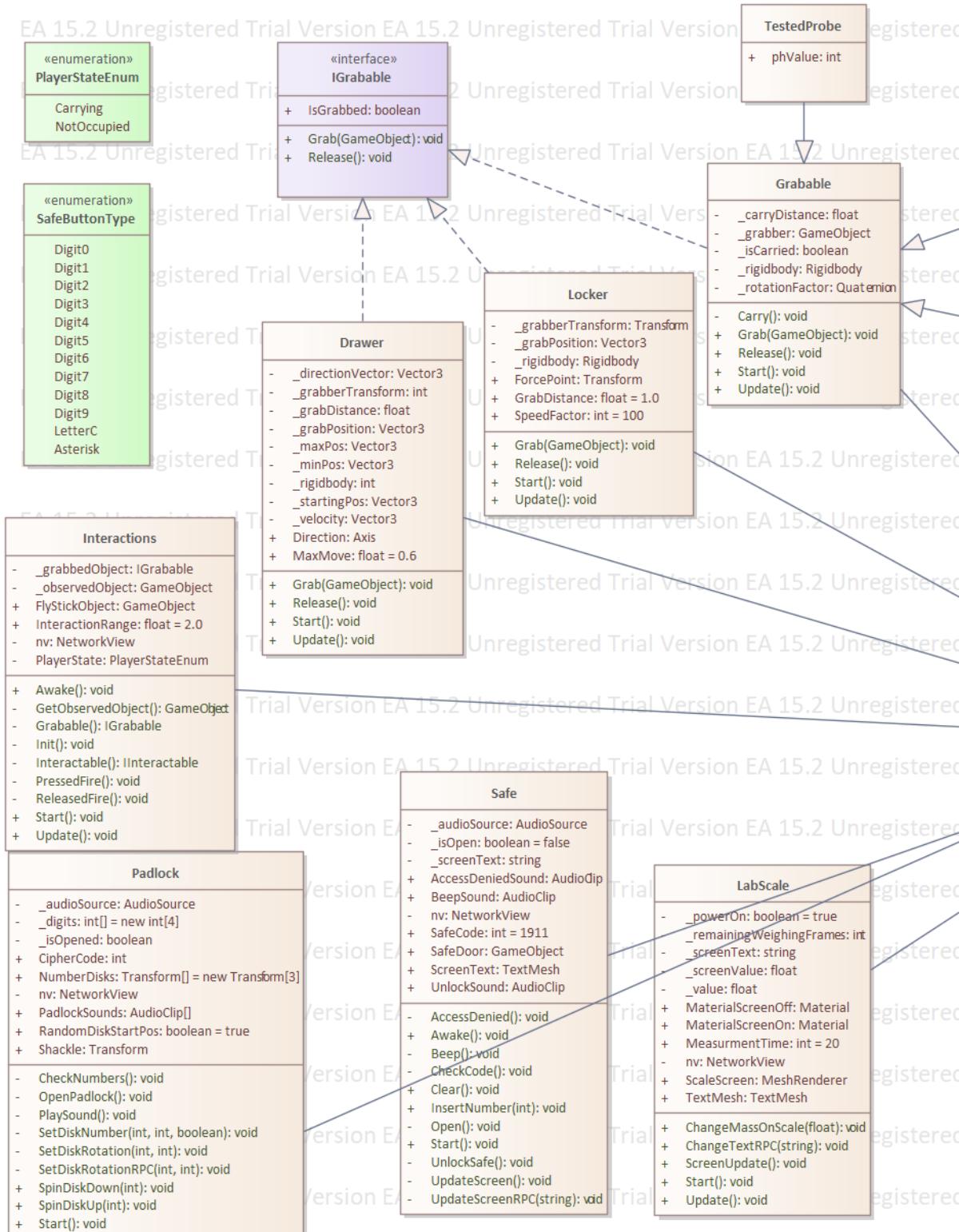
4. SPECYFIKACJA WYMAGAŃ (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)

Rozgrywka w aplikacji wymagała implementacji pewnych zachowań, dzięki którym immersja będzie silniejsza. Dodatkowo należało mieć na uwadze fakt, że gra będzie rozgrywana w specjalistycznym środowisku, Big Cave, co wymuszało dostępność określonych czynności.

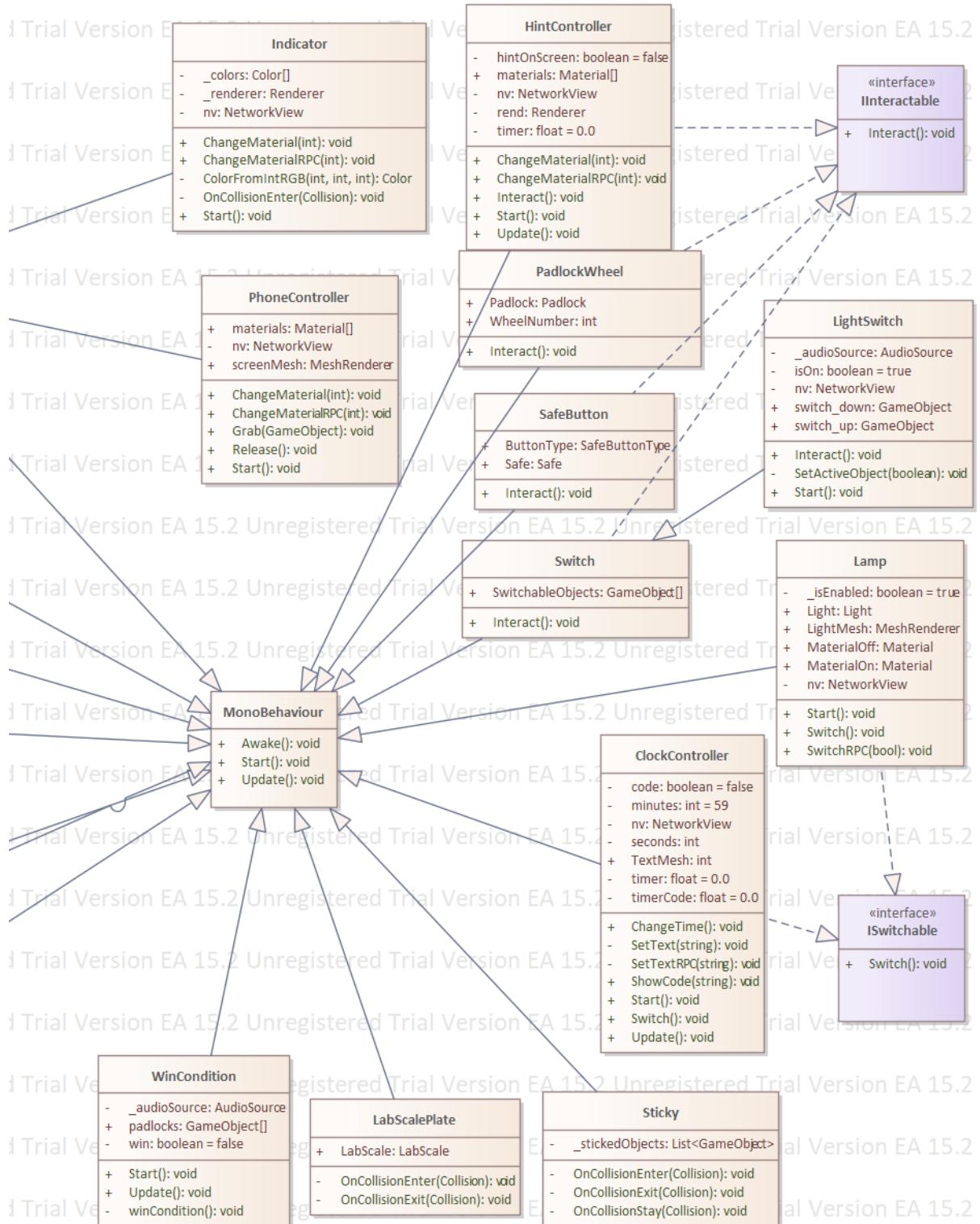
- Stworzenie wirtualnego pomieszczenia odpowiadającego rozmiarami jaskini BigCAVE (3.4 m x 3.4 m x 3.4 m).
- Zachowanie rzeczywistej skali obiektów (1 m odpowiada 1 jednostce długości w *Unity*).
- Zaimplementowanie mechanizmów przemieszczania i manipulacji obiektem.
- Zachowanie rotacji chwytyanych przedmiotów względem użytkownika.
- Możliwość przemieszczania się po wirtualnym pokoju.
- Kłódki oraz sejf otwierają się po wpisaniu właściwego kodu.
- Projektowane zagadki są rozwiązywalne w kilku krokach.
- Pokój zawiera niezbędne przedmioty do rozwiązania zagadek.
- Wirtualny pokój wypełniony jest przedmiotami umożliwiającymi wchodzenie w interakcję i zabawę.
- Dostępny jest system podpowiedzi dla gracza.

4.1. *Diagram klas*

Implementacyjny diagram klas użytych w aplikacji przedstawiony został na rys. 4.1, 4.2.



Rysunek 4.1: Implementacyjny diagram klas cz.I



Rysunek 4.2: Implementacyjny diagram klas cz. II

5. IMPLEMENTACJA

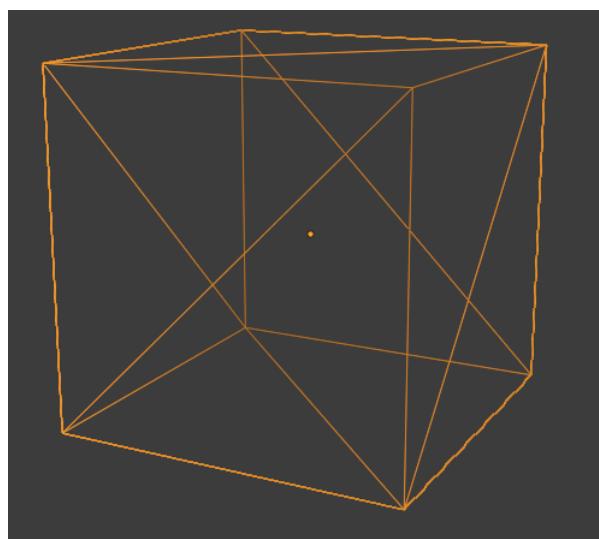
5.1. Modelowanie 3D (Przemysław Reszka)

Jednym z kluczowych elementów podczas tworzenia gry jest odwzorowanie świata poprzez użycie możliwie realistycznych modeli. Modelowanie jest dość żmudnym i czasochłonnym procesem, jednak wywiera istotny wpływ na wrażenia z rozgrywki. Należy przy tym pamiętać, że nadmierna szczegółowość tworzonych obiektów może negatywnie wpływać na wydajność aplikacji. Istotne jest tutaj również zastosowanie odpowiedniej skali oraz proporcji względem pozostałych obiektów.

Istnieje wiele sposobów na reprezentację obiektów trójwymiarowych, jednak jedynym wykorzystanym przy tworzeniu gry był ten za pomocą siatki wielokątów. Siatkę taką możemy tworzyć manipulując odpowiednio wierzchołkami, które to połączone tworząć będą krawędzie, a te z kolei możemy wypełnić ścianami [4].

5.1.1. Triangulacja

Jedną z podstawowych zasad przy modelowaniu jest unikanie wielokątów mających więcej niż cztery wierzchołki. Wynika to z faktu, że akceleratory graficzne operują na trójkątach, na które to zostaną rozłożone przygotowywane modele. Gwarantuje to również, że wierzchołki będą znajdować się w jednej płaszczyźnie. W przypadku czworokąta podział taki jest dość prosty — jest to podział jedną z dwóch możliwych przekątnych (rys. 5.1). Gdy kątów w wielokącie jest więcej proces nie jest już taki prosty [1].



Rysunek 5.1: Triangulacja sześcianu.

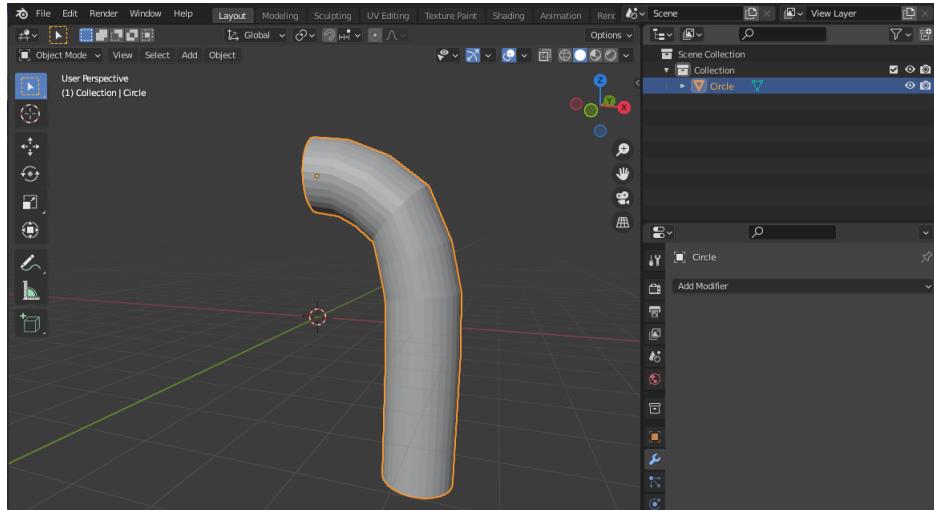
Niektórzy graficy aby osiągnąć taki efekt od początku modelowania posługują się trójkątami. Ta metoda może być jednak zbyt uciążliwa. Łatwiejszym sposobem jest zastosowanie triangulacji, czyli techniki polegającej na rozbiciu złożonych powierzchni bryły na trójkąty. W kontrolowanych warunkach programu np. *Blenderze* z modyfikatorem *Triangulate*, możemy dokonać takiego procesu z zachowaniem interesującej nas geometrii obiektu.

5.1.2. Modyfikatory

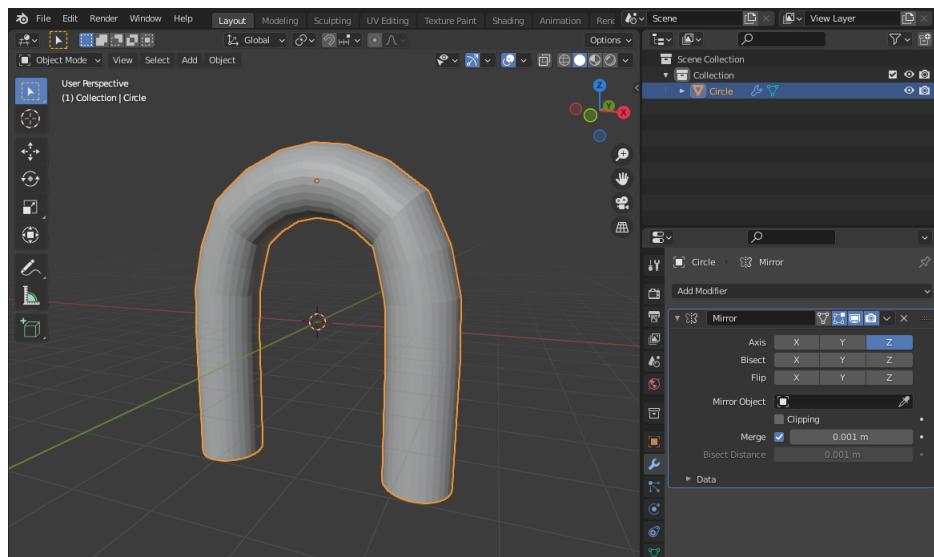
W trakcie modelowania wiele operacji wykonywanych na obiekcie nie różni się od siebie i możliwa jest ich automatyzacja. Blender umożliwia to poprzez użycie modyfikatorów. Umiejętnie wykorzystanie niektórych z nich pozwoliło na znaczne przyspieszenie długiego procesu modelowania [8].

Mirror

Jest to jeden z podstawowych modyfikatorów. Pozwala on na odbicie wierzchołków obiektu względem jednej bądź kilku osi. Znajduje więc zastosowanie w każdym modelu, gdzie zauważymy, że ma on przynajmniej jedną oś symetrii (rys. 5.2, 5.3).



Rysunek 5.2: Obiekt przed zastosowaniem modyfikatora Mirror.

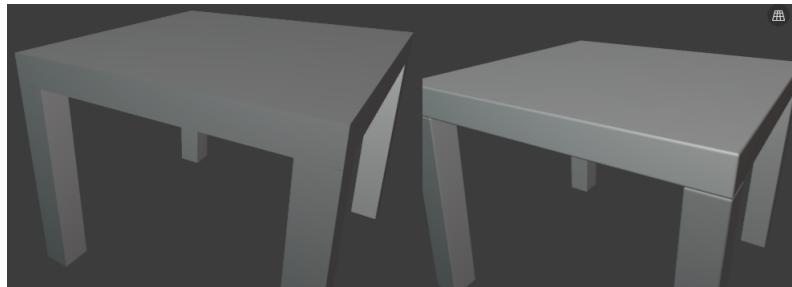


Rysunek 5.3: Obiekt po zastosowaniu modyfikatora Mirror.

Oprócz opcji wyboru osi odbicia modyfikator pozwala na przekształcenia względem innego dowolnego obiektu. Na uwagę również zasługuje opcja *Clamping*, która zapobiega przemieszczeniu wierzchołków w trybie edycji przez płaszczyznę odbicia. Z kolei włączenie opcji *Merge* połączy wierzchołki z ich lustrzanymi odpowiednikami o ile znajdą się one od siebie we wskazanej odległości.

Bevel

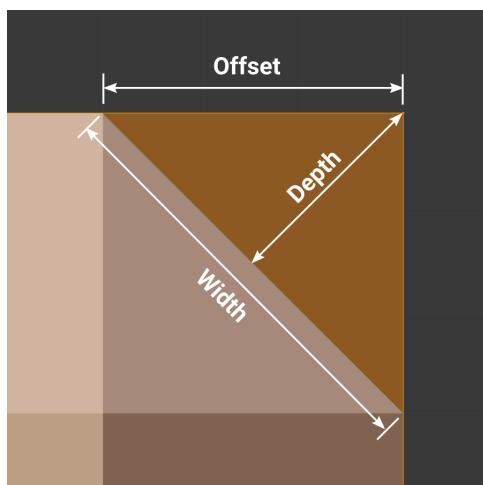
Również bardzo popularnym modyfikatorem, użytym przy tworzeniu większości modeli, jest *Bevel*. Służy on w uproszczeniu do zaokrąglania krawędzi poprzez ich powiększenie. Pozwala na unrealistycznie modelu dzięki nieskomplikowanej zasadzie mówiącej, że w rzeczywistości nic nie jest idealne, a krawędzie nigdy nie są perfekcyjnie ostre. Szczególnie jest to widoczne dla kątów prostych np. rogów ścian, krawędzi mebli czy blatów (rys. 5.4).



Rysunek 5.4: Urealistycznenie prostego modelu stołu za pomocą modyfikatora *Bevel*.

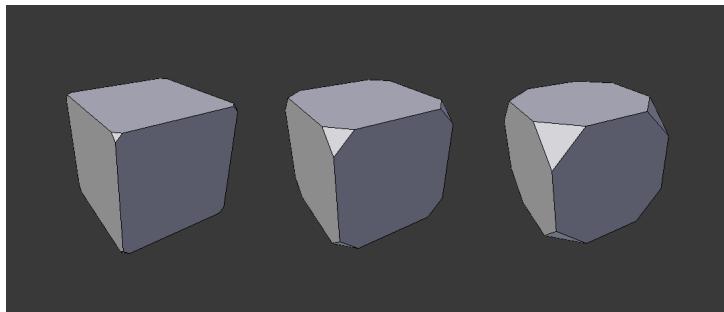
Korzystając z modyfikatora ustalamy szerokość oraz ilość segmentów podziału. *Bevel* oferuje kilka typów, które definiują, jak będzie interpretowana ta szerokość przy obliczaniu skosu:

- *Offset* — odległość od nowej krawędzi do oryginalnej (rys. 5.5).
- *Width* — odległość między dwiema nowymi krawędziami (lub krawędziami po obu stronach skosu jeśli jest więcej niż jeden segment).
- *Depth* — wartość ta jest prostopadłą odległością od nowej powierzchni skosu do oryginalnej krawędzi (rys. 5.5).
- *Percent* — procent długości przyległych krawędzi, po których przesuwają się nowe krawędzie.
- *Absolute* — dokładna odległość krawędzi przylegających do krawędzi fazowanej. Różnica w stosunku do *Offset* jest widoczna, gdy krawędzie przymocowane do ukośnych krawędzi spotykają się pod kątem innym niż kąt prosty.



Rysunek 5.5: Główne typy interpretacji szerokości w pracy modyfikatora *Bevel*.

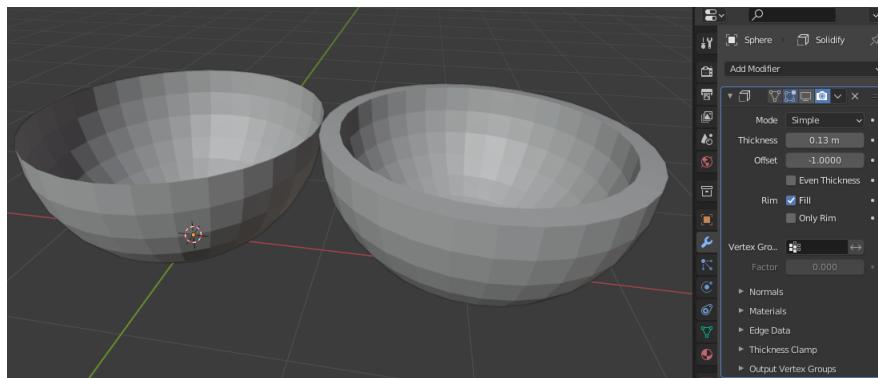
Modyfikator posiada również tryb pracy dla wierzchołków (rys. 5.6) oraz wiele dodatkowych opcji dla definiowania profilu tworzonych skosów.



Rysunek 5.6: Sześciany z *Bevel* w trybie wierzchołków dla wartości 0.1, 0.3, 0.5.

Solidify

Kolejnym użytym modyfikatorem jest *Solidify*. Służy on nadawaniu grubości obiektom, dzięki temu możemy pracować nad samą płaską siatką obiektu nie martwiąc się o grubość ścian (rys. 5.7). Jedyne co należy wówczas zrobić to podanie żądanej grubości (*Thickness*) oraz kierunku jej nadania (*Offset*). Posiada on dwa tryby — *Simple* — dla obiektów o prostej geometrii, wystarczający w większości przypadków, oraz *Complex* — który umożliwia dodatkową konfigurację dla brył bardziej złożonych.



Rysunek 5.7: Proste wykorzystanie modyfikatora *Solidify*.

Boolean

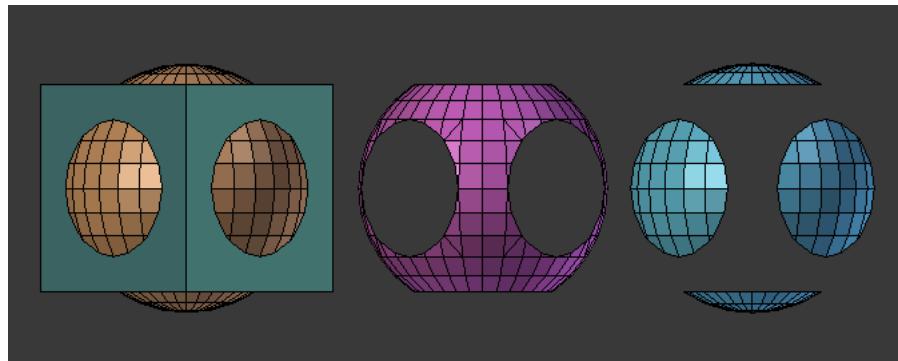
Do bardzo przydatnych modyfikatorów należy także *Boolean* wykonujący operacje na siatkach, które ciężko byłoby wykonać w przypadku ręcznej edycji. Wymaga on wskazania drugiego obiektu do transformacji i udostępnia trzy tryby pracy (rys. 5.8):

- *Union* — siatka wskazanego obiektu jest dodawana do obiektu modyfikowanego, usuwając wszystkie wewnętrzne ściany,
- *Intersect* — zachowywana jest część wspólna siatek obu obiektów,
- *Difference* — siatka wskazanego obiektu jest odejmowana od obiektu modyfikowanego (wszystko poza siatką obiektu wskazanego jest zachowywane).

5.1.3. Eksport modeli

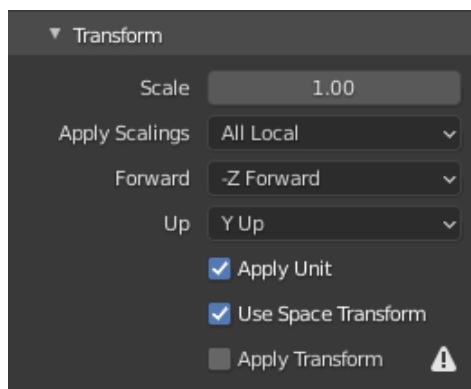
Powstałe w *Blenderze* modele zapisywane w plikach o rozszerzeniu .blend są gotowe do przeniesienia do silników gier. W przypadku *Unity* zostaną one automatycznie przekonwertowane na pliki .fbx. FBX jest to format zapisu plików graficznych 2D lub 3D, służący do zapewnienia interoperacyjności między aplikacjami do tworzenia treści cyfrowych.

Czasami w przypadku niekompatybilności wersji *Unity* i plików .blend może się pojawić błąd automatycznej konwersji, dlatego wszystkie przygotowane w projekcie modele zostały ręcznie przekonwertowane do formatu .fbx za pomocą wbudowanego narzędzia w *Blenderze*.



Rysunek 5.8: Operacje *Union*, *Intersection* oraz *Difference* na sferze, gdzie sześcian jest obiektem wskazanym przy operacji.

Blender oraz *Unity* posiadają różne modele osi – w *Unity* w góre jest skierowana oś Y, w *Blenderze* zaś jest to oś Z, dlatego przy konwersji zastosowano odpowiednie ustawienia transformacji zalecane przez społeczność twórców *Unity* (rys. 5.9).



Rysunek 5.9: Zastosowane ustawienia eksportu do pliku .fbx.

5.1.4. Utworzone modele

Poniższa lista przedstawia wszystkie modele znajdujące się w projekcie wraz z krótkim opisem oraz obrazem przedstawiającym dany model po dodaniu do niego materiałów.

Wszystkie modele zostały wykonane samodzielnie przy użyciu programu *Blender* w wersji 2.92.

1. light_switch

Przedstawia przełącznik do światła (rys. 5.10). Możliwa jest z nim interakcja, powodująca zmianę położenia przełącznika, zgaszenie światła lub pokazanie kodu na zegarze. Na scenie znajdują się 3 takie przedmioty.



Rysunek 5.10: Przełącznik światła.

2. safe

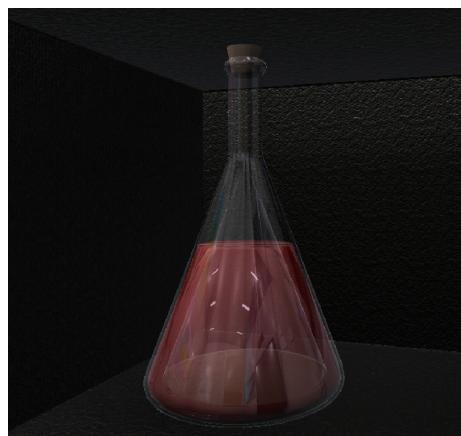
Przedstawia sejf (rys. 5.11). Możliwa jest z nim interakcja poprzez naciskanie cyfr na przednim panelu. Po wpisaniu poprawnego kodu interakcja zostaje zmieniona na możliwość otwarcia drzwi.



Rysunek 5.11: Sejf.

3. antidotum

Przedstawia antidotum, czyli szklaną fiolkę z czerwoną cieczą w środku oraz korkiem zamkającym naczynie (rys. 5.12). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.



Rysunek 5.12: Antidotum.

4. tube_rack

Przedstawia statyw na probówki (rys. 5.13). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.

5. padlock

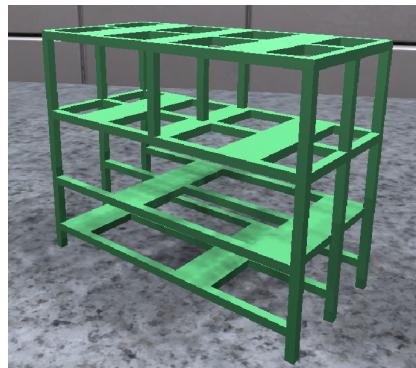
Przedstawia kłódkę (rys. 5.14). Istnieje możliwość interakcji, polegającej na przekręcaniu korby przedmiotu w celu ustawienia kodu. Po odblokowaniu interakcja zostaje zamieniona na podnoszenie i poruszanie się z przedmiotem. Na scenie znajduje się 5 kłódek.

6. corkboard

Przedstawia tablicę korkową (rys. 5.15). Istnieje możliwość interakcji, polegającej na przyczepianiu do niej kartek z napisanymi związkami chemicznymi.

7. ph_scale

Przedstawia plakat z ilustracją skali pH wraz z kolorami i opowiadającymi im numerami (rys. 5.16).



Rysunek 5.13: Statyw na próbówki.



Rysunek 5.14: Kłódka.



Rysunek 5.15: Tablica korkowa.

8. clock

Przedstawia zegar cyfrowy odliczający pozostały czas rozgrywki (rys. 5.17).

9. smartphone

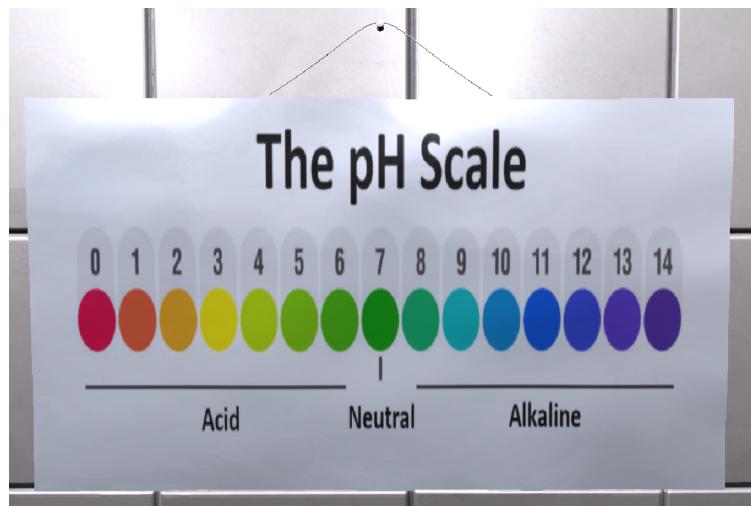
Przedstawia telefon komórkowy typu smartphone (rys. 5.18). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem. Gdy telefon jest trzymany pojawia się na nim ekran czatu tekstowego, natomiast w stanie spoczynku pokazuje czarny, czyli wyłączone, ekran.

10. tv

Przedstawia telewizor domyślnie wyświetlający nazwę pokoju zagadek (rys. 5.19). Istnieje opcja interakcji, polegająca na zmianie wyświetlanego obrazu na podpowiedź do losowej zagadki, która pozostaje na ekranie przez 30 sekund (rys. 5.20).

11. mendeleev_board

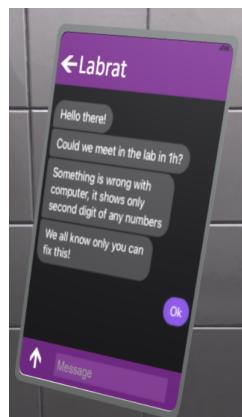
Przedstawia układ okresowy pierwiastków (rys. 5.21).



Rysunek 5.16: Skala pH.



Rysunek 5.17: Zegar cyfrowy.



Rysunek 5.18: Smartfon.

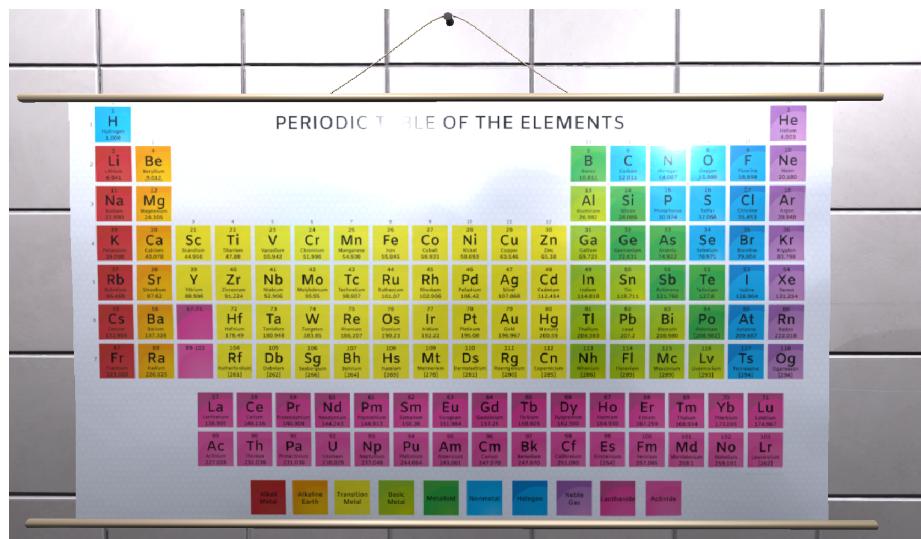
12. book
Przedstawia książkę (rys. 5.22). Dostępne są 4 warianty (blue, green, red, brown), różniące się jedynie kolorem okładki. Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.
13. lab_scale
Przedstawia wagę (rys. 5.23). Istnieje możliwość interakcji polegającej na włączeniu i wyłączeniu przedmiotu. Gdy położy się inny przedmiot na wagę, ta pokaże jego ciężar.
14. lab_table
Przedstawia stół z szufladami oraz szafkami (rys. 5.24). Istnieje możliwość interakcji polegającej na wysuwaniu, wsuwaniu szuflad oraz otwieraniu, zamknięciu szafek.
15. lab_table2
Przedstawia drugi stół (rys. 5.25).



Rysunek 5.19: Telewizor wyświetlający ekran domyślny.



Rysunek 5.20: Telewizor wyświetlający ekran z podpowiedzią.



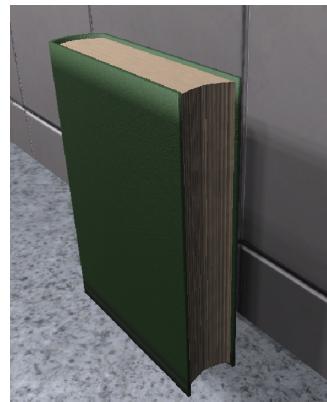
Rysunek 5.21: Układ okresowy pierwiastków.

16. pictures

Przedstawiają zdjęcia chemików wraz z datami otrzymania przez nich pierwszej Nagrody Nobla (rys. 5.26). Na scenie znajduje się 5 obrazów (picture_zigimondy, picture_buchner, picture_skłodowska, picture_joliot, picture_howard).

17. door

Przedstawia drzwi, framugę, uchwyt oraz wrzeciądze do kłódek (rys. 5.27).



Rysunek 5.22: Książka.



Rysunek 5.23: Waga.



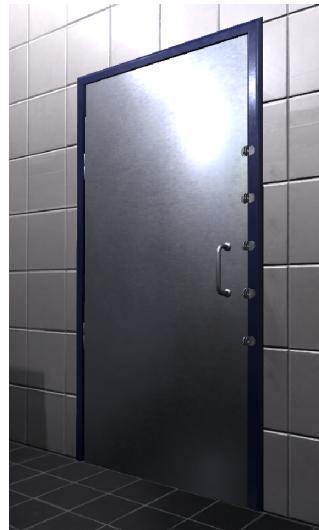
Rysunek 5.24: Stół w szufladami i szafkami.



Rysunek 5.25: Drugi stół.



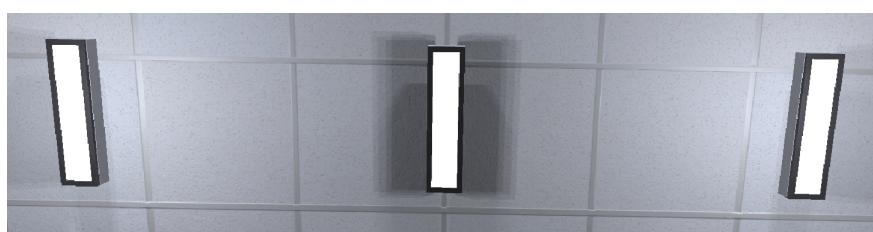
Rysunek 5.26: Zdjęcia chemików.



Rysunek 5.27: Drzwi.

18. ceiling_lamp

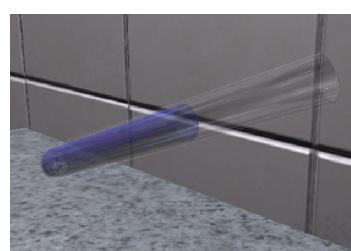
Przedstawia lampa, która jest zawieszona na suficie (rys. 5.28). Przedmiot ten jest źródłem światła na scenie. W aplikacji znajduje się 6 lamp.



Rysunek 5.28: Lampy.

19. tube

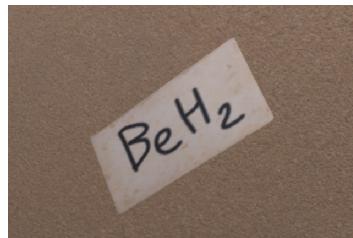
Przedstawia próbówkę (rys. 5.29). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.



Rysunek 5.29: Probówka z niebieską cieczą.

20. compound

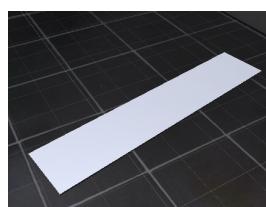
Przedstawia kartkę papieru z napisanym związkiem chemicznym (rys. 5.30). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem oraz przyczepienie go do tablicy korkowej. Na scenie znajduje się 5 kartek.



Rysunek 5.30: Kartka z napisanym związkiem chemicznym przyczepiona do tablicy korkowej.

21. paper_indicator

Przedstawia papier wskaźnikowy (rys. 5.31, 5.32). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.



Rysunek 5.31: Papier wskaźnikowy.



Rysunek 5.32: Papierki wskaźnikowe po zetknięciu z probówkami.

22. office_bin

Przedstawia śmietnik (rys. 5.33).



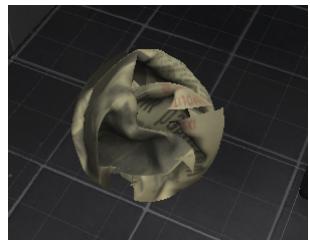
Rysunek 5.33: Śmietnik.

23. thrash

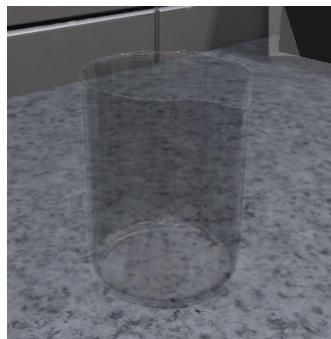
Przedstawia śmieć (rys. 5.34). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.

24. beaker

Przedstawia zlewkę (rys. 5.35). Możliwe jest podnoszenie i przemieszczanie się z przedmiotem.



Rysunek 5.34: Śmieć.



Rysunek 5.35: Zlewka

25. wall

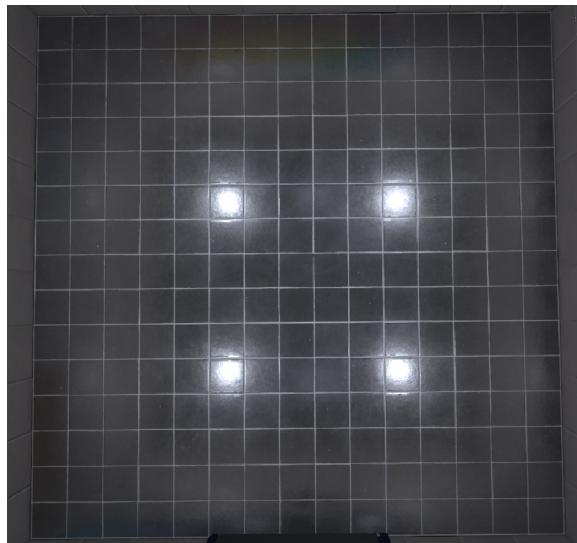
Przedstawia ścianę (rys. 5.36). Na scenie znajdują się 4 ściany.



Rysunek 5.36: Ściana.

26. floor

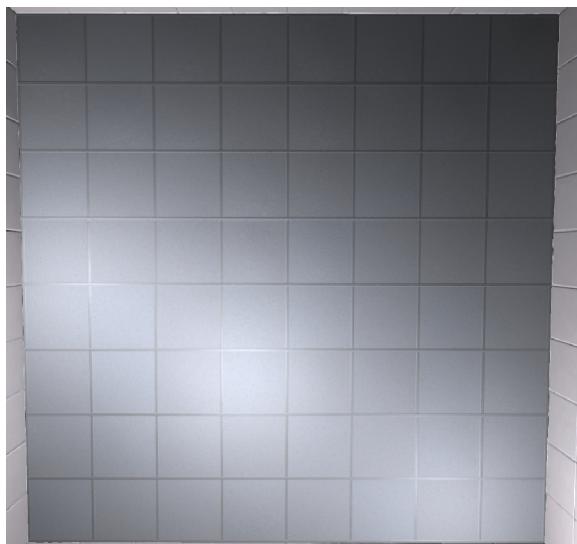
Przedstawia podłogę (rys. 5.37).



Rysunek 5.37: Podłoga.

27. ceiling

Przedstawia sufit (rys. 5.38).



Rysunek 5.38: Sufit.

5.2. Komponenty i interakcje

Po importie obiektu do *Unity* mamy możliwość dodania go do sceny. Początkowo jest on statycznym obiektem zawieszonym w przestrzeni. Jednak aby aplikacja spełniała funkcję gry obiektom znajdującym się na scenie należy nadać pewne właściwości, co można osiągnąć za pomocą komponentów [7].

5.2.1. Komponenty Unity (Przemysław Reszka)

Komponenty są to skompilowane moduły programowe, nadające cechy i zachowania obiektom do których zostaną przypisane. Ich pracę możemy kontrolować za pomocą ich publicznych parametrów.

Inicjalnie każdy model dodany do sceny posiada 3 komponenty:

- *Transform* — który przechowuje informacje o położeniu, rotacji oraz skali obiektu,
- *Mesh filter* — posiadający referencję do siatki danego obiektu,

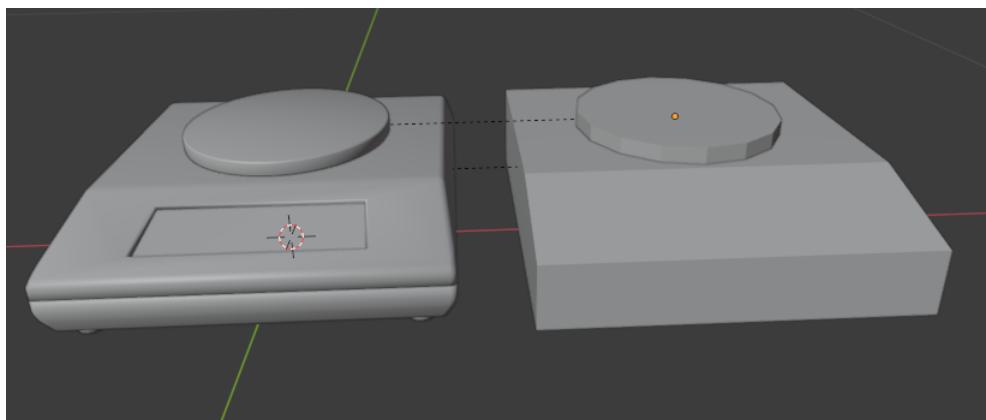
- *Mesh Renderer* — renderer obiektu, który definiuje sposób w jaki ma być wyświetlany obiekt na scenie.

Unity zapewnia również bazę komponentów gotowych do użycia. Do najpopularniejszych należy *Rigidbody*. Dodanie komponentu *Rigidbody* spowoduje oddanie kontroli nad ruchem obiektu silnikowi fizycznemu *Unity*. Bez zamieszczania żadnego dodatkowego kodu obiekt będzie ciągnięty w dół przez grawitację oraz będzie reagował na kolizje z innymi obiektami (jeżeli posiada *Collider*). *Rigidbody* posiada również wygodne API skryptowe, które pozwala na nakładanie sił na obiekt i sterowanie nim w realistyczny fizycznie sposób.

Do obsługi kolizji *Rigidbody* wymaga, aby obiekt miał nadany dowolny komponent *Collider*. Komponent ten odpowiedzialny jest za nadanie tzw. kolidera obiektem, który przekazuje silnikowi informacje o zajściu kolizji obiektu. *Unity* posiada kilka prostych koliderów dla popularnych kształtów m.in. sześciangu (*Box Collider*), kapsułki (*Capsule Collider*), kuli (*Sphere Collider*). Możliwa jest edycja tego kolidera np. zmiana rozmiaru lub położenia względem naszego obiektu.

Oprócz standardowych koliderów dostępny jest również *Mesh Collider* który pozwala na użycie własnej bryły np. siatki obiektu, do której dodany jest komponent. W użyciu tego komponentu wraz z komponentem *Rigidbody* występuje jednak pewne ograniczenie. Mianowicie *Rigidbody* wymaga do obsługi kolizji, aby użyty kolider był bryłą wypukłą.

Ze względu na kosztowność obliczeń związanych z kolizjami zaleca się, aby w przypadku użycia *Mesh Collider* dla złożonych modeli o dużej liczbie wierzchołków wykorzystać uproszczony kolider tzn. wersję tego samego modelu o zredukowanej złożoności i ilości użytych wierzchołków (rys. 5.39).



Rysunek 5.39: Użycie uproszczonego kolidera dla modelu wagi.

Użycie komponentów *Unity* pozwala na utworzenie pewnych ram projektowych, jednak zaprogramowanie działań poszczególnych przedmiotów i ich interakcji w procesie rozwiązywania zagadek wymagało napisania dedykowanych skryptów.

5.2.2. Klasa *MonoBehaviour* (Mikołaj Trylewicz)

Każdy obiekt znajdujący się na scenie (wraz z nią) jest obiektem klasy *GameObject*. Aby zdefiniować czym dokładnie jest dany obiekt należy dodać do niego komponenty, takie jak kolidery, przekształcenia czy skrypty. Skrypty w dużej mierze definiują zachowanie się danego obiektu, jednak żeby na niego zadziałały, o ile są napisane w języku C#, muszą dziedziczyć po klasie *MonoBehaviour* [3].

Jest to podstawowa klasa programu *Unity*, zawierająca wszystkie metody i zmienne potrzebne do prawidłowego działania skryptów definiujących zachowania obiektów na scenie. Nie oznacza to jednak, że wszystkie skrypty w projekcie muszą ją zawierać.

Oferuje ona metody zarządzające kolejnością wywołania, o ile są zawarte w skrypcie, takie jak [5]:

- *Awake()* – zostaje wywołana w pierwszej kolejności i tylko raz w trakcie ładowania instancji skryptu;
- *Start()* – zostaje wywołana tylko raz, zaraz przed pierwszym wywołaniem metody *Update()*. Najczęściej służy ona za miejsce, w którym przypisuje się zmiennym komponenty;
- *Update()* – zostaje wywoływana raz na klatkę. W niej znajduje się kod opisujący zachowania obiektów na scenie, takie jak zmiana materiału czy poruszanie się;
- *LateUpdate()* – zostaje wywołana raz na klatkę, ale po wywołaniu wszystkich funkcji *Update()*.

5.2.3. Interfejsy (Przemysław Reszka)

Zastosowanie interfejsów pozwoliło na wygodną obsługę akcji obiektów w *Interactions.cs* bez ich dokładnej znajomości.

Zaplanowane interakcje zachodzące przy rozwiązywaniu zagadek podzielone zostały ze względu na swoją specyfikę na trzy grupy reprezentowane przez oddzielne interfejsy.

IGrabable

Pierwsza z nich reprezentowana przez interfejs *IGrabable* odpowiada za akcje związane z chwytyaniem, przemieszczaniem czy rotacją obiektów np. otwieranie szuflad i szafek, podniesienie telefonu.

```
interface IGrabable
{
    bool IsGrabbed { get; set; }
    void Grab(GameObject grabber);
    void Release();
}
```

Interfejs zawiera metody *Grab* oraz *Release* odpowiedzialne odpowiednio za chwycenie i wypuszczenie przedmiotu. Metoda *Grab* jako argument przyjmuje *GameObject* obiektu chwytającego. Udostępnia również pole *IsGrabbed* reprezentujące status chwycenia obiektu implementującego interfejs.

IInteractable

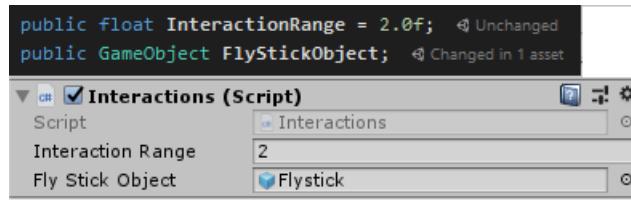
Drugą grupą są interakcje polegające na wywołaniu własnych akcji obiektu, bez modyfikowania jego własności. Reprezentowane są przez interfejs *IInteractable* zapewniający wyłącznie jedną bezparametrową metodę *Interact*.

```
public interface IInteractable
{
    void Interact();
}
```

ISwitchable

Interfejs *ISwitchable* podobny do *IInteractable* różni się od niego tym, że metody klas go implementujących wywoływane nie będą bezpośrednio przez interakcję gracza, a przez inne obiekty np. w przypadku interakcji z przełącznikiem który wywoła metodę *Switch* implementowaną przez klasę lampy czy zegara.

```
public interface ISwitchable{
    void Switch();
}
```



Rysunek 5.40: Zastosowanie publicznych pól w komponencie.

5.2.4. Interakcje gracza (Przemysław Reszka)

Skryptem odpowiadającym za obsługę interakcji gracza jest *Interactions.cs*. Na scenie został on umieszczony na pustym elemencie w roli *Game Managera*, czyli pewnego rodzaju konfigurowalnego zarządcy gry.

Pojawienie się edytowalnych pól wyświetlanych w komponencie w edytorze *Unity* (rys. 5.40), pozwalających na wygodną konfigurację działania skryptu zachodzi na skutek użycia publicznych pól klasy.

Główną rolą skryptu *Interactions.cs* jest obsługa zdarzenia interakcji gracza z otoczeniem, wywoływanej naciśnięciem lub zwolnieniem przycisku na kontrolerze *FlyStick*.

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.L))
    {
        if (PlayerState==PlayerState.Carrying)
            ReleasedFire();
        else if (PlayerState==PlayerState.NotOccupied)
            PressedFire();
    }
}
```

Powyższy fragment kodu przedstawia, wymaganą w przypadku uruchamiania na komputerze osobistym, symulację naciskania i zwalniania przycisku kontrolera za pomocą pojedynczych naciśnień przycisku na klawiaturze z zachowaniem stanu gracza.

Przedstawiona niżej metoda *PressedFire()* odpowiada za realizację akcji przy naciśnięciu przycisku kontrolera.

```
private void PressedFire()
{
    _observedObject = GetObservedObject();
    var grabable = Grabable(_observedObject);
    var interactable = Interactable(_observedObject);
    if (interactable != null) interactable.Interact();
    if (grabable == null) return;
    grabable.Grab(FlyStickObject);

    PlayerState = PlayerState.Carrying;
    _grabbedObject = grabable;
}
```

Na początku metody pobierany jest *GameObject* obserwowanego obiektu za pomocą metody *GetObservedObject()* i zapisywany w prywatnej zmiennej. Następnie pobierane są interfejsy *IGrabable* oraz

IInteractable danego obiektu po czym wywoływana jest ich obsługa. W przypadku *IGrabable* konieczne jest odpowiednie ustawienie aktualnego stanu gracza reprezentowanego przez *PlayerStateEnum*

```
private enum PlayerStateEnum
{
    Carrying,
    NotOccupied
}
```

Poniżej przedstawiono kod metody *GetObservedObject()* służącej do pozyskania wskazywanego obiektu.

```
GameObject GetObservedObject()
{
    RaycastHit hit;
    if (Physics.Raycast(FlyStickObject.transform.position,
        FlyStickObject.transform.forward, out hit))
    {
        if (hit.distance < InteractionRange)
            return hit.collider.gameObject;
    }
    return null;
}
```

Do znalezienia obiektu posłużono się metodą *Physics.Raycast*, która bada, w co trafia promień (półprosta) zaczynający się w podanym punkcie oraz zmierzający w podanym kierunku. Jako punkt początkowy wskazana została pozycja obiektu przypisanego do publicznego pola *FlyStickObject*, a jako kierunek podany został specjalny wektor *forward* (który jest zawierany przez każdy obiekt *Transform*), oznaczający oś Z danego obiektu.

Metoda *Raycast* zwraca *true* jeśli nastąpiło trafienie w obiekt zawierający kolider, a informacje o trafieniu zapisuje w zmiennej *hit* typu *RaycastHit*. Obiekt klasy *RaycastHit* posiada wiele przydatnych właściwości m.in. podaje dokładną odległość obiektu od punktu początkowego, którą porównujemy z ustawioną maksymalną odlegością interakcji *InteractionRange*.

Poniższe fragmenty kodu służą do pozyskiwania interfejsów *IGrabable* oraz *IInteractable* z przekazanych do nich obiektów. Wykorzystana w nich metoda *GetComponent<>()* klasy *GameObject* zwraca wskazany komponent przypisany do obiektu. Jeżeli obiekt takiego nie posiada, zwracany jest *null*. Użycie tej metody podając interfejs sprawi, że otrzymamy dowolną klasę implementującą ten interfejs.

Przed zwróceniem interfejsu następuje dodatkowo sprawdzenie, czy przypisany komponent jest włączony za pomocą pola *enabled* klasy *MonoBehaviour*. Takie rzutowanie jest możliwe ze względu na to, że każdy komponent musi dziedziczyć po *MonoBehaviour*, a więc klasy implementujące szukane interfejsy również.

```
private IGrabable Grabable(GameObject gameObject)
{
    var grabable = gameObject.GetComponent<IGrabable>();
    return grabable != null && ((MonoBehaviour) grabable).enabled ?
        grabable : null;
}
private IInteractable Interactable(GameObject gameObject)
```

```

{
    var interactable = gameObject.GetComponent<IInteractable>();
    return interactable != null && ((MonoBehaviour) interactable).enabled ?
        interactable : null;
}

```

Zamieszczona niżej metoda *ReleasedFire()* realizuje akcje po zwolnieniu przycisku kontrolera *FlyStick*.

```

private void ReleasedFire()
{
    if (_grabbedObject == null) return;
    _grabbedObject.Release();
    _grabbedObject = null;
    PlayerState = PlayerStateEnum.NotOccupied;
}

```

Jeżeli trzymany jest aktualnie jakiś obiekt wywoływana jest metoda *Release* interfejsu *IGrabable*. Następnie pole jest zerowane i ustawiany jest odpowiedni stan gracza *PlayerState*.

5.2.5. Klasy *IGrabable* (Przemysław Reszka)

Grabable

Podstawową klasą implementującą interfejs *IGrabable* jest klasa *Grabable*. Reprezentuje obiekty o najczęściej występującej w pokoju zagadek interakcji tj. możliwości ich przenoszenia.

```

protected void Start()
{
    _rigidbody = GetComponent<Rigidbody>();
    IsGrabbed = false;
}

```

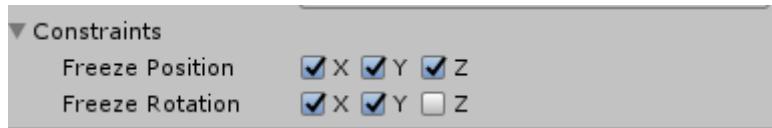
W metodzie *Start* następuje przygotowanie obiektu czyli pobranie i zapisanie w prywatnej zmiennej obiektu *Rigidbody* oraz ustawienie flagi *IsGrabbed* na fałsz.

W metodzie *Grab* implementowanego interfejsu zapisywany jest do prywatnej zmiennej obiekt 'chwytający' oraz ustawiana jest flaga *IsGrabbed*. Następnie ustawiane są pola *useGravity* oraz *angularDrag*. Pierwsze z nich odpowiada za sygnalizowanie, czy na obiekt ma oddziaływać grawitacja, a drugi - za wartość oporu kątowego obiektu. Zapisywana jest również odległość od chwyconego obiektu będąca długością wektora wynikowego z różnicą pozycji obiektu chwytanego i chwytającego.

```

public virtual void Grab(GameObject grabber)
{
    IsGrabbed = true;
    _grabber = grabber;
    _rigidbody.useGravity = false;
    _rigidbody.angularDrag = 10.0f;
    _carryDistance = (transform.position - grabber.transform.position)
        .magnitude;
    _rotationFactor = Quaternion
        .Inverse(_grabber.transform.rotation) * transform.rotation;
}

```



Rysunek 5.41: Wartości Constraints komponentu Rigidbody.

Zapisywany jest również do zmiennej `_rotationFactor` współczynnik rotacji będący relacją rotacji chwytnego obiektu do obiektu chwytającego. Wartość ta będzie potrzebna do zachowania stałej rotacji względem gracza podczas przenoszenia przedmiotu.

W metodzie `Update` wywoływanej przez *Unity* co każdą klatkę, jeżeli podniesiona jest flaga `IsGrabbed`, wywoływana jest funkcja `Carry` odpowiedzialna za obliczenie nowej pozycji i rotacji obiektu.

```
void Update ()
{
    if (IsGrabbed) Carry ();
}

private void Carry ()
{
    _rigidbody.velocity = (_grabber.transform.position +
        _grabber.transform.forward * _carryDistance - transform.position)
        * CarrySpeed;
    transform.rotation = _grabber.transform.rotation * _rotationFactor;
}
```

Wykonanie przemieszczenia obiektu realizowane jest przez nadanie odpowiedniej prędkości komponentowi *Rigidbody*. Prędkość ta obliczana jest z różnicy aktualnego położenia obiektu od punktu będącego docelową pozycją obiektu tj. punktem leżącym na osi Z obiektu trzymanego i oddalonym o odległość początkową `_carryDistance`. Dodatkowo wartość jest pomnożona przez współczynnik prędkości `CarrySpeed`.

Następnie obliczana jest wartość rotacji obiektu z wartości rotacji obiektu trzymanego pomnożonej przez współczynnik rotacji `_rotationFactor`.

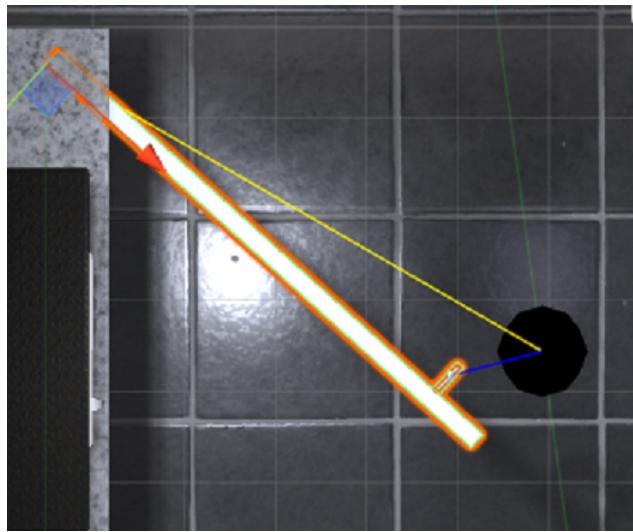
Locker

Klasa *Locker* opisuje zachowanie otwieranych szafek. Takie zachowanie znacznie różni się od opisywanego wyżej *Grabable* — obiekt nie będzie zmieniał swojego położenia, a jedynie rotację w tylko jednej z osi. W tym celu w komponencie *Rigidbody* (rys. 5.41) zostały zablokowane wszystkie osie ruchu oraz rotacji poza jedną wybraną.

W metodzie `Start` dodatkowo został ustawiony środek ciężkości komponentu *Rigidbody* na lokalną pozycję danego obiektu będącą współrzędnymi jego *PivotPointa*. Sprawi to że rotacja obiektu będzie odbywała się względem tego punktu, a nie geometrycznego środka obiektu.

W celu poprawnej rotacji obiektu ustawiana jest także rotacja tensora bezwładności na zerową. Domyslnie wartość obliczana jest na podstawie dzieci przypisanych do danego obiektu na scenie.

```
void Start ()
{
    _rigidbody = GetComponent<Rigidbody>();
    _rigidbody.centerOfMass = transform.localPosition;
    _rigidbody.inertiaTensorRotation = Quaternion.identity;
```



Rysunek 5.42: Użycie Gizmos w edytorze Unity.

}

W metodzie *Update* w przypadku podniesionej flagi *IsGrabbed* odbywa się ruch obiektu. Najpierw obliczany jest 'punkt chwytu' tzn. punkt leżący na przedłużeniu osi Z oddalony o odległość z momentu chwycenia. Następnie przykładany jest obliczony wektor siły w punkcie *ForcePoint* będącym współrzędnymi obiektu klamki.

```
void Update ()
{
    if (IsGrabbed)
    {
        _grabPosition = _grabberTransform.position +
            _grabberTransform.forward * GrabDistance;
        _rigidbody.AddForceAtPosition(( _grabPosition - ForcePoint.position )
            * SpeedFactor, ForcePoint.position);
    }
}
```

W trakcie opracowywania mechanizmu szczególnie pomocne było użycie klasy *Gizmos*, pozwalającej za pomocą metody *OnDrawGizmos()* na rysowanie kształtów w oknie edytora. Umożliwiło to wizualizację obliczanych wektorów ruchu.

Użycie *Gizmos* widoczne jest na rys. 5.42, gdzie czarny punkt odpowiada pozycji *_grabPosition*, linia niebieska – wektorowi siły, a linia żółta – docelowej rotacji obiektu.

Drawer

Klasa *Drawer* opisuje działanie szuflad. Opisywany w niej sposób interakcji niewiele różni się od klasy *Grabable*. Jedyną różnicą jest przeskalowanie nadawanego wektora prędkości tak, aby ruch odbywał się tylko w jednej osi.

Ponadto komponent posiada możliwość ograniczenia ruchu obiektu w osi wskazanej przez publiczne pole *Direction* typu wyliczeniowego *Axis*.

```
void Start ()
{
    _rigidbody = GetComponent<Rigidbody>();
```

```

        _rigidbody.drag = 1.0f;
        _startingPos = transform.position;
        switch (Direction)
        {
            case Axis.None:
                throw new Exception("Select the axis of movement");
                break;
            case Axis.X:
                _directionVector=Vector3.right;
                break;
            case Axis.Y:
                _directionVector=Vector3.up;
                break;
            case Axis.Z:
                _directionVector=Vector3.back;
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
        _maxPos = _startingPos + _directionVector * MaxMove;
        _minPos=_startingPos + _directionVector * MaxMove *(-1);
    }
}

```

W metodzie *Start* odwzorowana zostaje oś na odpowiedni wektor, który posłuży do przeskalowania ruchu. Obliczone zostają również punkty maksymalnego wychylenia na wybranej osi o wskazaną wartość *MaxMove*.

PhoneController

Dziedzicząca po *Grabable* klasa *PhoneController* definiuje zachowanie telefonu komórkowego.

```

public override void Grab(GameObject grabber)
{
    ChangeMaterial(1);
    base.Grab(grabber);
}

public override void Release()
{
    ChangeMaterial(0);
    base.Release();
}

```

Klasa ta nadpisuje odziedziczone metody *Grab* i *Release*, aby przy podnoszeniu i opuszczaniu obiektu nastąpiła zmiana materiału, a następnie wywołanie bazowych metod.

5.2.6. Klasy *IInteractable* (Mikołaj Trylewicz)

HintController

Działanie telewizora jest opisane w skrypcie *HintController.cs*. Zawiera on metody, definiujące co znajduje się na ekranie. Domyślnie ekran wyświetla nazwę aplikacji. Po wykryciu interakcji gracza z

ekranem zostaje wylosowana liczba z przedziału [1, 6], która reprezentuje indeks w tablicy zawierającej materiały z podpowiedziami do zagadek. Wylosowany materiał zostaje podmieniony na ekranie telewizora i wyświetla się przez 30 sekund. Po upływie tego czasu materiał zostaje podmieniony na domyślny.

SafeButtons

Działanie przycisków znajdujących się na sejfie definiuje skrypt *SafeButtons.cs*. Zawiera on publiczną zmienną typu *Safe*, która odnosi się do obiektu sejfu na scenie, oraz publiczną zmienną typu *SafeButtonType*, która jest typem wyliczeniowym zdefiniowanym w skrypcie i jego elementami są wszystkie przyciski znajdujące się na drzwach sejfu. Po wykryciu interakcji gracza z przyciskiem wywoływane są odpowiednie metody klasy *Safe.cs*. Gdy naciśniętym przyciskiem był przycisk C zostaje wywołana metoda *Clear()*, która czyści ekran sejfu. Natomiast jeśli został naciśnięty przycisk odpowiadający cyfrze, zostaje wywołana metoda *InsterNumber(int)*, która dodaje do ekranu wybraną cyfrę.

PadlockWheel

Działanie korbek znajdujących się na kłódkach definiuje skrypt *PadlockWheel.cs*. Zawiera ona publiczną zmienną typu *int* odpowiadającą numerowi korbki (pierwsza od góry ma numer 0, druga 1, trzecia 2), oraz publiczną zmienną typu *Padlock*, która odnosi się do obiektu kłódki na scenie. Po wykryciu interakcji gracza z daną korbką zostaje wywołana metoda klasy *Padlock.cs* o nazwie *SpinDiskUp(int)*, która odpowiedzialna jest za animacje przekręcenia korbki w prawo.

Switch

Klasa *Switch.cs* jest specjalnie utworzonym skryptem, dzięki któremu po interakcji z obiektem go dziedziczącym zachodzą zmiany w wybranych obiektach dziedziczących po interfejsie *ISwitchable*. Zawiera ona publiczną tablicę typu *GameObject*, która przechowuje obiekty, na których ma być wywołana zmiana (w przypadku projektu są to lampy i zegar). Jedyna metoda o nazwie *Interact()* jest metodą wirtualną, aby klasy dziedziczące mogły ją nadpisać. Aktywuje ona w każdym obiekcie w tablicy metodę *Switch()*.

LightSwitch

Działanie przełączników światła opisane jest w skrypcie *LightSwitch.cs*. Klasa ta dziedziczy po klasie *Switch*, zatem ma możliwość nadpisania metody *Interact()*. Nadpisana metoda zmienia wygląd przełącznika na włączony lub wyłączony (w zależności w jakim stanie się znajduje) oraz wywołuje bazową metodę *Interact()*.

ClockController

Działanie zegara jest opisane w skrypcie *ClockController.cs*. Zawiera on metody, dzięki którym wyświetlają się odpowiednie cyfry na ekranie. Domyślnie obliczany i wyświetlany jest czas, który pozostał do zakończenia rozgrywki. Dostępna jest również metoda o nazwie *Switch()*, która wyświetla na ekranie kod do jednej z kłódek przez 5 sekund, jednocześnie nie zakłócając obliczeń upływającego czasu rozgrywki. Metoda ta zostaje wywołana jedynie po wejściu w interakcję z odpowiednim właczniem światła.

Lamp

Działanie lampy opisane jest w skrypcie *Lamp.cs*. Posiada on metodę *Switch()*, wyłącza lub włącza komponent *light* i ustawia odpowiedni do sytuacji materiał lampy. Metoda ta zostaje wywołana jedynie po wejściu w interakcję z odpowiednim włacznikiem światła.

5.2.7. Klasy dziedziczące tylko po MonoBehaviour (Mikołaj Trylewicz)

WinCondition

Klasa *WinCondition.cs* jest odpowiedzialna za sprawdzanie, czy wszystkie kłódki zostały odblokowane. Gdy tak się stanie, odtwarzany jest dźwięk, oznajmiający wygranie gry.

Indicator

Klasa *Indicator.cs* jest odpowiedzialna za wykrycie kolizji patyczka wskaźnikowego z probówką i zmienienie jego koloru. Zawiera zmienną typu *GameObject*, która reprezentuje dany patyczek oraz zmienną typu *Material* reprezentującą materiał, który zostanie zmieniony na patyczku. Skrypt jest docelowo ustawiony na probówce.

LabScale

Działanie wagi jest opisane w skryptach *LabScale.cs* oraz *LabScalePlate.cs*. Pierwszy z nich zawiera metody odpowiadające za wyświetlanie odpowiednich cyfr na ekranie oraz ich animację. Drugi odpowiada za interakcję przedmiotu z szalką. Gdy przedmiot zostanie na niej położony, zostaje wywołana metoda należąca do pierwszego skryptu odpowiadająca za przypisanie wartości ciężaru przedmiotu do zmiennej, która następnie jest wyświetlana na ekranie. Gdy przedmiot zostanie usunięty z szalki, znów zostaje wywołana ta sama metoda z pierwszego skryptu, jednak przesyłana wartość równa jest zeru.

Padlock

Działanie kłódki jest opisane w skrypcie *Padlock.cs*. Na początku w metodzie *Start()* zostają ustawione pozycje wszystkich korbek. Po każdej interakcji z jedną z nich, za którą odpowiada skrypt *PadlockWheel.cs*, zostaje ona przekręcona w prawo, słyszać odpowiedni dźwięk oraz zostaje sprawdzany kod, który reprezentuje położenie korbek. Gdy kod zgadza się z ustawionym wcześniej kodem, kłódka zostaje odblokowana, czyli zostaje dodany do obiektu komponent *Grabbable*. W tym momencie zostaje również wyłączona możliwość ustawiania kodu, ale za to kłódkę można podnosić i przenosić w dowolne miejsce.

Safe

Działanie sejfu opisane jest w skrypcie *Safe.cs*. Zawiera on metody odpowiadające wpisywaniu na ekran cyfr znajdujących się na drzwiach, czyszczeniu ekranu po naciśnięciu przycisku C (interakcje przycisków zdefiniowane są w skrypcie *SafeButtons.cs*), wyświetleniu komunikatu o niepoprawnie wpisanym kodzie, odtwarzaniu odpowiednich dźwięków, sprawdzaniu poprawności kodu oraz zdarzeniu po odblokowaniu sejfu. Gdy zostanie wpisany prawidłowy kod, do obiektu reprezentującego drzwi sejfu zostaje dodany komponent *Locker*, dzięki któremu będzie można otworzyć sejf, tak jak w przypadku szafki znajdującej się w biurku.

Sticky

Działanie tablicy korkowej opisane jest w skrypcie *Sticky.cs*. Posiada on prywatną listę typu *GameObject*, do której zostają dodawane obiekty posiadające tag *sticky* (pol. lepki) podczas wykrycia kolizji z tablicą. Gdy kolizja dalej ma miejsce, po upuszczeniu przez gracza obiektu, obiektowi zostaje wyłączona grawitacja oraz prędkości liniowa i kątowa są zmienione na zero. Dzięki temu obiekt powoduje wrażenie przyczepionego do tablicy. Gdy gracz chwyci zawieszony obiekt i nie będzie już on kolidował z tablicą, zostanie on usunięty z listy.

5.3. Tekstury i materiały (Mikołaj Trylewicz)

Wszystkie tekstury znajdujące się w projekcie zostały albo wykonane samodzielnie poprzez ustawienie odpowiednich kolorów, wykonane samodzielnie w programie GIMP w wersji 2.10, albo zostały pobrane z różnych stron internetowych oferujących tekstury z darmową licencją.

Materiały do poszczególnych modeli są tworzone na podstawie tekstur. Definiują one tylko podstawowy wygląd materiału, inne opcje takie jak *shader* czy *rendering mode* można ustawić w oknie *Inspector* materiału w programie Unity.

Poniższy spis materiałów dostępnych w aplikacji został podzielony ze względu na tryb renderingu. W każdym przypadku stosowany shader jest shaderem standardowym, zatem nie wystąpiła konieczność dodatkowego podzielenia spisu wg tego atrybutu.

5.3.1. Materiały emisyjne

W przypadku opisywanej aplikacji, materiały z zaznaczoną opcją *Emmision* imitują emisję światła przez ekranów różnych urządzeń elektrycznych (tab. 5.1).

Tabela 5.1: Materiały emisyjne

Nazwa	Modele	Tekstury
lamp_light	ceiling_lamp	lamp_light
lcd_screen_on	lab_scale, Safe	lcd_screen_on
lcd_screen_off	clock, smartphone	lcd_screen_off
lcd_screen_red	clock	lcd_screen_red
screen	tv	screen
hint1-6	tv	hint1-6
phone	smartphone	phone

5.3.2. Materiały transparentne

Używane są do modeli, które mają reprezentować szkło lub ciecz. Dzięki opcji *transparent* modele są przeźroczyste, zatem można zobaczyć co znajduje się za nimi (tab. 5.2).

Tabela 5.2: Materiały transparentne

Nazwa	Modele	Tekstury
fluid_blue	tube	fluid_blue
fluid_pink	tube	fluid_pink
fluid_red	tube, antidotum	fluid_red
fluid_yellow	tube	fluid_yellow
glass	tube, antidotum, picture, beaker	glass

5.3.3. Materiały zanikające

Dzięki opcji *Fade* w *Rendering Mode* materiał poprawnie przechwytuje przeźroczystość z tekstyurą i nakłada ją na model (tab. 5.3).

Tabela 5.3: Materiały zanikające

Nazwa	Modele	Tekstyury
backside_blank	Compound1-5	backside_blank
compound1-5	Compound1-5	compound1-5

5.3.4. Materiały zwykłe

Podstawowa opcja renderingu, materiały są trwałe bez różnych urozmaiceń (tab. 5.4).

Niektóre materiały różnią się jedynie wariantami kolorystycznymi ustawnionymi w inspektorze, dla tego dla uproszczenia warianty te zostały uwzględnione w nawiasach przy danym materiale.

Większość tekstur użytych w tych materiałach ma nazwę albedo, metallic, roughness, normal, height oraz occlusion i są one ustawiane w odpowiednich polach w oknie Inspector. Dlatego dla uproszczenia będą one nazywane odpowiednio a, m, r, n, h oraz o.

Tabela 5.4: Materiały zwykłe

Nazwa	Modele	Tekstury
damaged_rubber	lab_scale, lab_table	dr (a, r, n, h, o)
light_off	ceiling_lamp	light_off
leather (blue, brown, green, red)	book	leather (r, n, h)
aluminium_brushed	office_bin, safe, padlock, lab_scale, lab_table, door	alu (a, r, n)
brass	padlock	brass (a, r, n)
grain_painted_metal	safe	gpm (n, r)
grinded_steel	door	gs (a, r, n, h)
metal_hammered	safe	mh (a, r, n, h, m, o)
painted_metal (normal, blue, grey)	safe, padlock, clock, door, lab_table, lab_table2	pm (a, m, r, n)
plain_steel	padlock,	ps(a, r, n)
office_ceiling	ceiling	oc (a, m, r, n, h, o)
book_pages	book	bp (a, r, n, h)
cork	corkboard, antidotum	cork (a, r, n, h, o)
mandelev_board	mandelev_board	mandelev_board, mb_normal
ph_scale	ph_scale	ph_scale, ph_normal
photo_frame	picture_(buchner, howard, joliot, sklodowska, zsigmondy)	pf (a, m, n, r)
buchner	picture_buchner	buchner
howard	picture Howard	howard
joliot	picture_joliot	joliot
sklodowska	picture_sklodowska	sklodowska
zsigmondy	picture_zsigmondy	zsigmondy
plastic (black, grey)	ceiling_lamp	plastic (a, r, n)
scuffed_plastic (green, black, white)	tube_rack tv, lab_scale, lab_table2	scuffed-plastic (green, black, white, n, o)
plywood_clean	corkboard	plywood_albedo, plywood (r, n)
phone_material	smartphone	phone_material
tiles (dark, white)	floor, wall	tile (a, n, r, h, o)
white_granite_slab	lab_table, lab_table2	wgs (a, r, n)
indicator (0, 2, 7, 9)	indicator	indicator (0, 2, 7, 9)
trash (1, 2, 3, 4)	trash (1, 2, 3, 4)	trash_paper (1, 2, 3, 4)

6. URUCHOMIENIE APLIKACJI W ŚRODOWISKU CAVE (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)

Przeniesienie aplikacji do Laboratorium Zanurzonej Wizualizacji Przestrzennej wiąże się z dodaniem do projektu pakietu LZWPlib. Zawiera on dwie biblioteki DLL, które udostępniają funkcje odpowiadające za prawidłowe działanie aplikacji w środowisku CAVE oraz dodatkowe okno z ustawieniami edytora [6].

Aplikacje w CAVE'ie działają na klastrze komputerowym o wielu instancjach. Jedna z nich pełni rolę serwera, reszta rolę klientów. Serwer odpowiedzialny jest za całą logikę aplikacji, czyli przeprowadzanie obliczeń na danych wejściowych, takich jak położenie gracza w aplikacji, interakcje, działania fizyczne czy też zmienianie obrazów na scenie, oraz za aktualizowanie stanu aplikacji i synchronizowanie go z innymiinstancjami. Klienci natomiast odpowiadają za wygenerowanie obrazów wyświetlanego na ścianach jaskini. Zatem przesyłane dane przez serwer do klientów to dane, które zmieniają rezultat wizualny działania aplikacji [2].

6.1. Funkcje LZWPlib

Pierwszą czynnością, którą należy wykonać to dodanie odpowiednich obiektów, czy też prefabrykatów, z biblioteki do sceny gry. W przypadku niniejszego projektu są to:

- LZWPlib – prefabrykat zawierający skrypt *Lzwp.cs*, który jest głównym skryptem biblioteki. Odpowiada on za inicjalizację biblioteki oraz podczas inicjalizacji programu zostaje przeniesiony na pierwszy poziom hierarchii, wywołuje się na nim metoda *DontDestroyOnLoad*, przez co nie zostanie on usunięty przy zmianie lub przeładowaniu sceny.
- LzwpOrigin_M – prefabrykat zawierający skrypy *LzwpOrigin.cs*, *LzwpOriginGizmos.cs* oraz *MovementController.cs* wraz z wymaganymi komponentami (Rigidbody, Audio Source, Capsule Collider, Character Controller). Pierwszy skrypt reprezentuje położenie i orientację rzeczywistej jaskini w wirtualnym świecie aplikacji. Drugi umożliwia wyświetlenie w oknie sceny reprezentacji ekranów wybranej jaskini, płaszczyzn przeszkołów fizycznych oraz reprezentacji śledzonych obiektów. Trzeci odpowiada za poruszanie się po scenie.
- CameraContainer – prefabrykat, który zawiera dwa obiekty z kamerami przeznaczone dla lewego i prawego oka.
- Flystick – obiekt reprezentujący fizyczny flystick dostępny w Laboratorium Zanurzonej Wizualizacji Przestrzennej. W niniejszym projekcie używane są tylko dwa pola obiektu: *pose* (odpowiadające za śledzenie pozycji kontrolera) oraz *OnButtonChange* (odpowiadające za akcje wywołane poprzez zmianę stanu przycisku, w tym wypadku wykorzystany jest jeden typ wyliczeniowy *ButtonID - Fire*).

Aby obiekt był śledzony przez kamery dodany jest również skrypt *TrackedObject.cs*.

Wszystkie obiekty znajdujące się na scenie dodatkowo powinny zawierać komponent *NetworkView*, który służy do synchronizacji położenia w jaskini pomiędzy wszystkimi instancjami aplikacji. Aby synchronizacja przebiegała pomyślnie kod znajdujący się w obiektach z tym komponentem powinien zawierać metody *RPC* (ang. *remote procedure call*), które służą do zdalnego wywoływanego procedur.

6.2. Okno edytora LZWPlib

Zimportowanie pakietu LZWPlib w Unity automatycznie otwierało nowe okno narzędziowe o takiej samej nazwie jak pakiet. Pozwala ono np. włączyć aplikację w trybie *miniCAVE* bez konieczności odwiedzania Laboratorium Zanurzonej Wizualizacji Przestrzennej i testowanie działania aplikacji na własnym komputerze. Zawiera ono 4 zakładki:

1. Project settings – zawiera listę wymaganych i opcjonalnych ustawień projektu związanych z jego działaniem w środowisku CAVE wraz z polami do podpięcia obiektów konfiguracji. W przypadku realizowanego projektu wszystkie pola były ustawione automatycznie, jedynie ustawienie opcji *Config* w dziale *Startup config for TESTING* na *miniCAVE* (rys. 6.1).



Rysunek 6.1: Zakładka project settings.

2. App metadata – umożliwia wprowadzenie danych opisujących tworzoną aplikację. W przypadku realizowanego projektu nie była edytowana.
3. Input – podczas działania aplikacji w edytorze wyświetla dane odebrane od systemu śledzenia. W przypadku realizowanego projektu nie była edytowana.
4. Testing – zawiera opcje pomagające w testowaniu synchronizacji wielu instancji aplikacji. W przypadku realizowanego projektu zmieniono opcję *Config object* na *miniCAVE*, ustawiono ID instancji oraz wybrano opcję *Before other instances* w opcji *Run in editor also*. Dzięki takim ustawieniom po wybraniu opcji *Build and run* można było testować działanie aplikacji jak w prawdziwym CAVE'ie.

6.3. Zmiany w kodzie

Zmiany, które trzeba było dokonać w kodzie, aby aplikacja działała poprawnie w jaskini nie były duże. Najważniejszym elementem było zrozumienie jak dane instancje porozumiewają się między sobą. W przypadku tego projektu wszystkie obliczenia dzieją się na serwerze, klienci otrzymują tylko informacje, co należy wyświetlić na scenie.

Wykorzystanie biblioteki *LZWPlib* pierwszy raz zostało użyte w skrypcie *Interactions.cs*, ponieważ definiuje on interakcje z przedmiotami znajdującymi się na scenie. W aplikacji *Escape room - laboratory* tylko interakcje zmieniają wygląd sceny, wpływając na położenie obiektów znajdujących się na scenie

oraz zmianę ich materiałów. Zatem najwygodniej było zacząć dodatkową inicjalizację i definicję reagowania na przyciski właśnie w tym skrypcie. Metoda *Start()* wygląda następująco:

```
void Start ()
{
    Lzwp.AddAfterInitializedAction(Init);
}
```

Zawiera ona funkcję z pakietu LZWPlib *AddAfterInitializedAction(Init)*, która wykonuje metodę *Init* tuż po inicjalizacji biblioteki, czyli na samym początku działania programu. Metoda w niej zawarta jest zdefiniowana następująco:

```
private void Init()
{
    Lzwp.input.flysticks[0]
        .GetButton(Lzwplib.Flystick.ButtonID.Fire)
        .OnPress += PressedFire;
    Lzwp.input.flysticks[0]
        .GetButton(Lzwplib.Flystick.ButtonID.Fire)
        .OnRelease += ReleasedFire;

    if (!Lzwp.sync.isMaster)
    {
        foreach (var rg in FindObjectsOfType<Rigidbody>())
        {
            Destroy(rg);
        }
    }
}
```

Pierwsza część metody mówi, jaka metoda ma się wykonać, gdy gracz naciśnie i przytrzyma odpowiedni przycisk na flysticku oraz jak go zwolni. W tym przypadku są to metody *PressedFire* oraz *ReleasedFire*. Druga część natomiast niszczy wszystkim elementom znajdującym się na scenie komponent *rigidbody*, ale tylko dla instancji klienckich. Jest to bardzo ważny fragment kodu, ponieważ bez niego klient próbowałby liczyć zachowania fizyczne dla tych obiektów podczas interakcji, co doprowadziłoby do konfliktu obliczeń przez serwer i obiekt zachowywałby się nienaturalnie.

Inne zmiany w kodzie dotyczą zmian materiałów obiektów znajdujących się na scenie. Domyslnie nowe materiały zostałyby wyświetlane tylko na instancji serwera, więc efekt byłby niezauważalny podczas rozgrywki. Aby przesłać dane z serwera do klientów najpierw w danej klasie należy stworzyć zmienną typu *NetworkView* i w metodzie start przypisać do niej ten komponent:

```
private NetworkView nv;

void Start ()
{
    nv = GetComponent<NetworkView>();
}
```

Następnie należy użyć metody RPC, która jest metodą obiektu *NetworkView*. Wywoływana jest ona w metodzie odpowiedzialnej za zmianę wyświetlonego materiału. Metoda bazowa oraz RPC nie różnią się znacząco od siebie. Ustawia ona taki sam materiał, jednak nie w instancji serwera, a w instancjach

klienckich. Aby metoda była wywołana za pośrednictwem mechanizmu wywoływania zdalnego musi być oznaczona atrybutem *RPC*. Przykład wywołania można pokazać w skrypcie *ClockController.cs*:

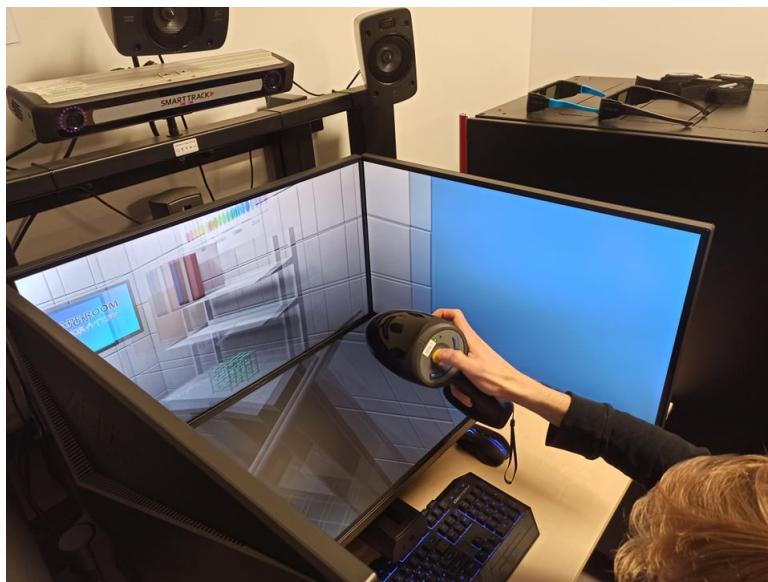
```
private void SetText(string text)
{
    TextMesh.text = text;
    nv.RPC("SetTextRPC", RPCMode.OthersBuffered, text);
}

[RPC]
private void SetTextRPC(string text)
{
    TextMesh.text = text;
}
```

Pierwszym argumentem metody *RPC* jest nazwa metody, która ma zostać wywołana. Drugim jest informacja, na jakich instancjach ma ona zostać wywołana (w podanym przykładzie są to inne instancje, niż ta, w której zostaje wywołana, czyli inne niż serwer). Ostatnim atrybutem jest zmienna, która zostanie przekazana do tych instancji.

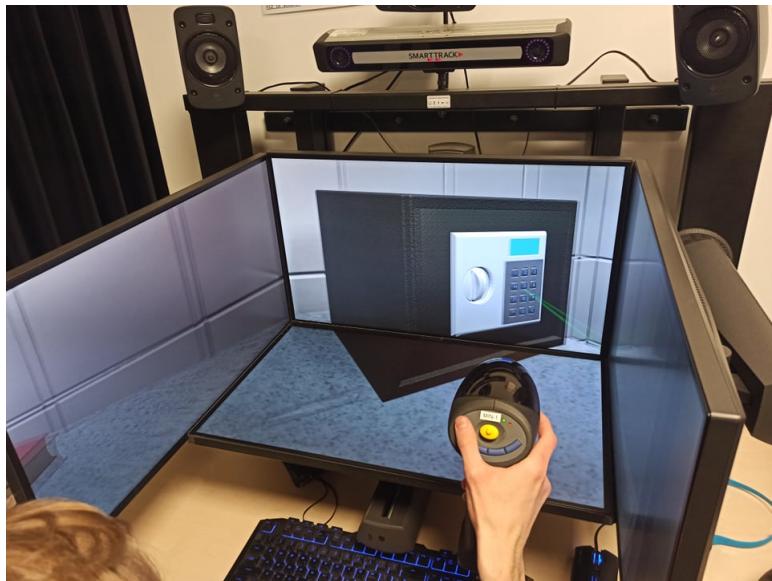
6.4. Testowanie aplikacji na miniCAVE

Podczas tworzenia aplikacji, szczególnie w procesie testowania zaimplementowanych funkcjonalności, wykorzystywane była uproszczona (względem BigCAVE) jaskinia rzeczywistości wirtualnej – MiniCAVE. Stanowisko składa się z czterech monitorów odpowiadających ekranowi dolnemu oraz trzem ekranom bocznym. Poruszanie się po takiej jaskini odbywa się za pomocą przycisków kontrolera Flystick (Rys. 6.2).



Rysunek 6.2: Poruszanie się w jaskini MiniCAVE.

Jaskinia swoimi rozmiarami nie przekracza standardowego stanowiska komputerowego co zapewniło wygodę przy użytkowaniu (Rys. 6.3).



Rysunek 6.3: Otwieranie sejfu na MiniCAVE.

6.5. Przykładowe przejście gry w BigCAVE’ie

Gracz po rozpoczęciu rozgrywki zaczął przeszukiwać szuflady. W jednej z nich znalazł telefon (rys. 6.4). Po przeczytaniu wiadomości zauważył, że pierwsze litery każdej wiadomości można powiązać z pierwiastkami znajdującymi się na tablicy Mendelejewa (rys 6.5). Po próbie rozwiązania szyfru gracz postanowił wpisać go w jednej z kłódek. Próba okazała się sukcesem i pierwsza kłódka została odblokowana (rys. 6.6, 6.7).

Przy okazji odblokowania kłódki gracz zauważył przy drzwiach włącznik światła. Interakcja z włącznikiem spowodowała zgaśnięcie 3 lamp (rys. 6.8). Gracz postanowił poszukać włączników, które zgaszą resztę lamp. Do kilku minutach odnalazł włącznik w dość nietypowym miejscu (rys. 6.9). Interakcja z nim nie skutkowała zgaszeniem się światel, zatem gracz zaczął rozglądać się po pokoju w celu ustalenia, co mogło się zmienić na scenie. Przez dłuższą chwilę nie mógł niczego zauważyc, więc postanowił znowu nacisnąć przycisk. Nagle zauważył, że zegar zamiast godziny wyświetla 4 cyfrową liczbę (rys. 6.10). Gracz był już pewien, że to jest kod do kolejnej kłódki.

Następną rzeczą, na którą gracz zwrócił uwagę, to kartki z napisanymi związkami chemicznymi (rys. 6.11). Zauważył również, że na scenie jest dostępna tablica korkowa. Postanowił więc znaleźć wszystkie kartki i spróbować przyczepić je do tablicy (rys. 6.12, 6.13). Po ułożeniu kartek był pewien, że muszą one być związanne z jedną z zagadek (rys. 6.14), tylko nie był pewny co z nimi zrobić. Postanowił więc zobaczyć co się stanie, jak wejdzie w interakcję z telewizorem, podejrzewając, że mogą wyświetlić się podpowiedzi. Niestety po pierwszej interakcji nie uzyskał podpowiedzi co należy zrobić z kartkami, ale dostał inną cenną informację, którą postanowił wykorzystać (rys. 6.15).

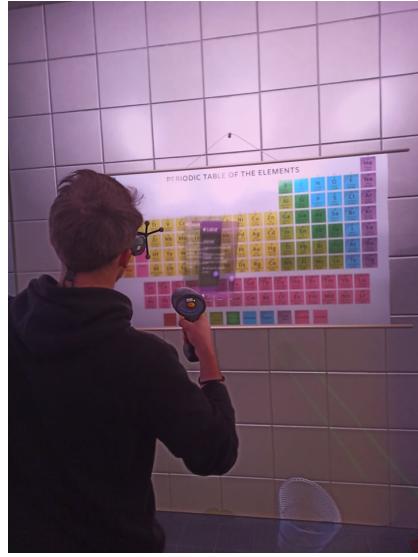
Gracz zauważył, że na scenie znajduje się kilka obrazów przedstawiających znanych chemików wraz z datami, które są złożone z 4 cyfr. Postanowił wpisać wszystkie daty do sejfu, ponieważ znajdował się on przy obrazach. Pierwsza próba nie okazała się sukcesem, więc gracz próbował wpisać inne daty (rys. 6.16, 6.17). W końcu udało się otworzyć sejf (rys. 6.18), w którym znajdowało się naczynie inne niż te, które można było znaleźć na scenie (rys. 6.19). Gracz postanowił położyć naczynie na wadze, ponieważ też znajdowała się blisko sejfu (rys. 6.20). Waga wyświetliła ciężar naczynia i gracz podejrzewał, że może to być kolejny kod do kłódki.

Gracz wiedząc, że pozostała mu tylko jedna zagadka do rozwiązania, postanowił zrobić coś z fiolkami porozrzucanymi po scenie (rys. 6.21). Zauważył również, że na drugim stole znajduje się statyw, do którego można włożyć fiołki (rys. 6.22, 6.23). Po prawidłowym ustawieniu fiolek w stojaku gracz zauważył, że kolory cieczy znajdujących się w fiołkach nie odpowiadają żadnym kolorom znajdującym się na skali pH (rys. 6.24). Postanowił zrobić coś z papierkami, które też były porozrzucane po scenie. Dotknięcie fiołki papierkiem wskaźnikowym skutkowało zmienieniem koloru papierka. Kolory te już pasowały do kolorów na skali pH (rys. 6.25, 6.26). Teraz wystarczyło tylko ułożyć papierki w odpowiedniej kolejności i odczytać kod (rys. 6.27).

Mając już wszystkie informacje gracz zaczął próbować ustawiać kody do kłódek (rys. 6.28). Po dłuższej chwili udało się odblokować wszystkie kłódki oraz wygrać rozgrywkę (rys. 6.29).



Rysunek 6.4: Odnaleziony telefon.



Rysunek 6.5: Próba rozwiązania zagadki z telefonem.

6.6. Poprawność działania aplikacji

Podczas testowych uruchomień aplikacji w jaskini BigCAVE zweryfikowana została poprawność działania zaimplementowanych mechanizmów rozgrywki, jakość zanurzenia oraz ogólna grywalność.



Rysunek 6.6: Próba ustawienia kodu do klódki.



Rysunek 6.7: Odblokowana klódka.



Rysunek 6.8: Zauważenie włącznika światła.



Rysunek 6.9: Odnalezienie drugiego włącznika.



Rysunek 6.10: Wyświetlany kod na zegarze.



Rysunek 6.11: Znalezienie pierwszej kartki.



Rysunek 6.12: Przykładanie kartek do tablicy korkowej.



Rysunek 6.13: Przenoszenie kartki na tablicę korkową.



Rysunek 6.14: Wszystkie kartki znajdują się na tablicy korkowej.



Rysunek 6.15: Wyświetlona podpowiedź.



Rysunek 6.16: Próba wpisania daty do sejfu.



Rysunek 6.17: Nieprawidłowy kod.



Rysunek 6.18: Prawidłowy kod.



Rysunek 6.19: Wnętrze sejfu.



Rysunek 6.20: Antidotum położone na wadze.



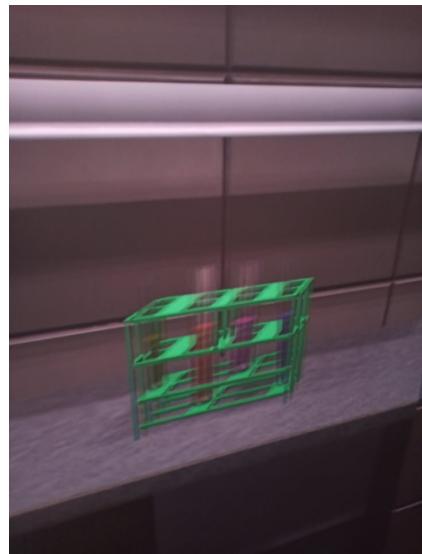
Rysunek 6.21: Odnalezienie jednej z fiolek.



Rysunek 6.22: Próba ustawiania fiolek w odpowiednie miejsce.



Rysunek 6.23: Dalsza próba ustawienia fiolek.



Rysunek 6.24: Poprawne ustawienie fiolek.



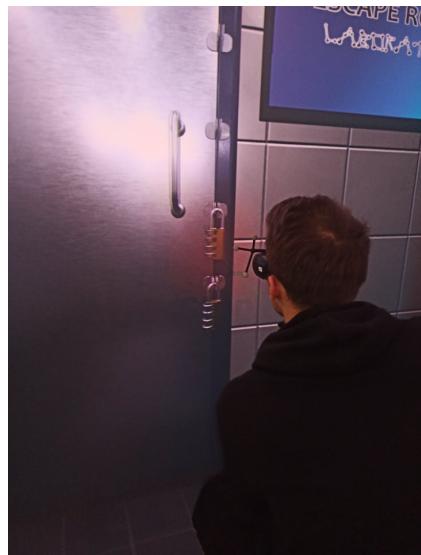
Rysunek 6.25: Porównywanie koloru na papierku z skalą pH.



Rysunek 6.26: Dalsza próba porównywania koloru.



Rysunek 6.27: Prawidłowe ustawienie papierków wskaźnikowych.



Rysunek 6.28: Próba odblokowywania reszty kłódek.



Rysunek 6.29: Wszystkie kłódkи zostały odblokowane.

6.6.1. Poprawnie działające mechanizmy

- Kłódki zostają odblokowane jedynie po ustawieniu poprawnego kodu i można je podnosić. W przeciwnym wypadku kłódka dalej jest przywieszona do wrzeciądza.
- Sejf zostaje odblokowany jedynie po wpisaniu poprawnego kodu, o czym informuje napis na ekranie "Open" oraz odpowiedni dźwięk. W przeciwnym wypadku na ekranie zostaje wyświetlony komunikat "Error" oraz słyszeć odpowiedni dźwięk.
- Wszystkie przedmioty zawierające komponent *Grabable* można swobodnie podnosić, przemieszczać się z nimi oraz je upuszczać.
- Przełączenie odpowiedniego przełącznika światła poprawnie wyświetla na zegarze kod do jednej z kłódek. Po upływie 5 sekund zegar znów wyświetla pozostały czas do ukończenia rozgrywki.
- Przełączenie odpowiednich przełączników światła poprawnie wyłącza lub włącza światło.
- Przeniesienie na wagę przedmiotu poprawnie pokazuje jego ciężar.
- Do tablicy korkowej można przyczepić jedynie kartki z napisanymi związkami chemicznymi. Inne przedmioty poprawnie upadają na ziemię przy próbie przyczepienia.
- Zegar poprawnie wyświetla pozostały czas do ukończenia rozgrywki.
- Przyłożenie patyczka wskaźnikowego do probówki poprawnie zmienia jego kolor.
- Po wejściu w interakcję z telewizorem poprawnie zmienia się treść na ekranie, po 30 sekundach automatycznie zostaje wyświetlany ekran początkowy.
- Telefon poprawnie zmienia wyświetlana treść na ekranie po podniesieniu go.
- Wyciąganie szuflad poprawnie blokuje się w odpowiednim miejscu.

6.6.2. Niepożądane działania

Czasami przedmioty, szczególnie te małe, przenikają przez kolidery innych obiektów. Widoczne jest to m.in. w przypadku szuflady i znajdujących się w niej obiektów. Również podczas szybkiego jej wysunięcia znajdujące się w niej przedmioty mogą przedostać się przez jej przednią część. Podobnie dzieje się w przypadku usilnego otwarcia szafki powyżej jej zakresu ruchu. Może to wynikać z błędów w posługiwaniu się komponentem *Rigidbody* lub z niedoskonałości silnika fizycznego Unity.

Mimo dwukrotnego powiększenia, względem rzeczywistych rozmiarów, obiektów kłódek wielokrotnie ustawianie kodu może stać się dla użytkownika uciążliwe z powodu w dalszym ciągu niewielkiego rozmiaru oraz trudności w operowaniu kontrolerem *Flystick*. Widoczne są również niedoskonałości śledzenia np. w przypadku przysłonięcia kontrolera ciałem. Uwidocznienie tych błędów może wynikać z konstrukcji niektórych zagadek — wielokrotnego wprowadzania kodów metodą prób i błędów.

Występują utrudnienia związane z interakcją na małych przedmiotach. Wiąże się to z specyfiką działania BigCAVE, co wymagało porzucenia rzeczywistej skali obiektów i dostosowania ich rozmiarów.

7. PODSUMOWANIE (PRZEMYSŁAW RESZKA, MIKOŁAJ TRYLEWICZ)

Celem powyższej pracy było zaprojektowanie wyglądu escape roomu, wymyślenie zagadek do rozwiązania, stworzenia aplikacji symulującej taką zabawę w świecie wirtualnym oraz umożliwienie działania jej w Laboratorium Zanurzonej Wizualizacji Przestrzennej znajdującym się na terenie Politechniki Gdańskiej, w budynku A Wydziału Elektroniki, Telekomunikacji i Informatyki.

Inspiracje do stworzenia fabuły oraz mechaniki zagadek zostały zaczerpnięte z istniejących już escape roomów, cyfrowych jak i tych w realnym świecie. Wieloletnie doświadczenie twórców w tym rodzaju rozgrywki zostało przeniesione do aplikacji z zadowalającym efektem.

Do wytworzenia aplikacji został użyty program Unity w wersji 2018.1.9, ponieważ wersja ta jest kompatybilna z pakietem LZWPLib, który zawiera potrzebne komponenty i funkcje do obsługi jaskini wirtualnej rzeczywistości. Program ten jest dosyć prosty w obsłudze, istnieje wiele samouczków w internecie na jego temat. Dodatkowo posiadane doświadczenie przez twórców pracy z nim wpływało pozytywnie na wytwarzanie aplikacji. Język programowania, który jest używany przez Unity to C#, zatem wszystkie skrypty potrzebne do działania aplikacji zostały napisane właśnie w tym języku.

Wszystkie modele dostępne na scenie zostały wykonane samodzielnie w programie Blender w wersji 2.92. Jedynie niektóre tekstury nakładane na modele zostały pobrane z internetu na darmowej licencji, ponieważ tworzenie specjalnych grafik jest czasochłonne, szczególnie dla osób z brakiem doświadczenia w tej dziedzinie.

Efektem końcowym jest aplikacja działająca poprawnie w jaskini wirtualnej rzeczywistości, dokładniej w pomieszczeniu o nazwie BigCAVE. Gra jest przystosowana dla jednej osoby, co nie znaczy, że w zabawie nie może uczestniczyć więcej osób, które mogą podpowiadać i razem rozwiązywać zagadki z graczem posiadającym potrzebny sprzęt. Gracz wchodząc do pomieszczenia zostaje wyposażony w specjalne okulary, których pozycje i ruch śledzą kamery, oraz we flystick, który pełni rolę kontrolera. Kamery śledzą i dostosowują wygląd sceny tylko do jednej pary okularów, zatem osoby posiadające inne okulary 3D mogą mieć problem z widocznością wszystkich elementów na scenie. Po uruchomieniu aplikacji gracz ma określony czas na rozwiązanie wszystkich zagadek i odblokowanie kłódek znajdujących się na wirtualnych drzwiach. W tym czasie może w pełni eksplorować pokój, wchodzić w interakcje z przedmiotami i badać mechanikę aplikacji. Uczestnik zabawy ma swobodę w rozwiązywaniu zagadek, nie ma żadnej przypisanej kolejności odblokowywania kłódek. Jednym przeciwnikiem jest czas oraz gracz we własnej osobie.

WYKAZ LITERATURY

- [1] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes: *Computer Graphics: Principles and Practice*, Second Edition. Addison-Wesley Reading 1990
- [2] J. Lebiedź: *Wyposażenie i zastosowania Laboratorium Zanurzonej Wizualizacji Przestrzennej. Elektronika –konstrukcje, technologie, zastosowania*, 7/2016
- [3] J. Ross: *Unity i C#. Praktyka programowania gier*. Helion 2020.
- [4] A. Thorn: *Unity i Blender. Praktyczne tworzenie gier*. Helion 2015
- [5] A. Thorn: *Mastering Unity 2017 Game Development with C#*, Second Edition. PACKT Publishing 2017
- [6] R. Trzosowski: *Interfejs programisty aplikacji dla jaskini rzeczywistości wirtualnej w Laboratorium Zanurzonej Wizualizacji Przestrzennej*. Praca magisterska, WETI 2020
- [7] Unity Documentation, <https://docs.unity3d.com/> (data dostępu 10.12.2021)
- [8] Blender Documentation, *Modifiers*, <https://docs.blender.org/manual/en/latest/modeling/modifiers.html> (data dostępu 10.12.2021)
- [9] The Escape Game, *The history of Escape Rooms* (2021), <https://theescapegame.com/blog/the-history-of-escape-rooms/> (data dostępu 10.12.2021)

WYKAZ RYSUNKÓW

2.1 Pokój zagadek - widok na drzwi.	9
2.2 Pokój zagadek - widok na układ okresowy pierwiastków.	10
2.3 Pokój zagadek - widok z góry.	10
4.1 Implementacyjny diagram klas cz.I	16
4.2 Implementacyjny diagram klas cz. II	17
5.1 Triangulacja sześcianu.	18
5.2 Obiekt przed zastosowaniem modyfikatora Mirror.	19
5.3 Obiekt po zastosowaniu modyfikatora Mirror.	19
5.4 Urealistycznenie prostego modelu stołu za pomocą modyfikatora <i>Bevel</i> .	20
5.5 Główne typy interpretacji szerokości w pracy modyfikatora <i>Bevel</i> .	20
5.6 Sześciany z <i>Bevel</i> w trybie wierzchołków dla wartości 0.1, 0.3, 0.5.	21
5.7 Proste wykorzystanie modyfikatora <i>Solidify</i> .	21
5.8 Operacje <i>Union</i> , <i>Intersection</i> oraz <i>Difference</i> na sferze, gdzie sześcian jest obiektem wskazanym przy operacji.	22
5.9 Zastosowane ustawienia eksportu do pliku .fbx.	22
5.10 Przelącznik światła.	22
5.11 Sejf.	23
5.12 Antidotum.	23
5.13 Statyw na probówki.	24
5.14 Kłódka.	24
5.15 Tablica korkowa.	24
5.16 Skala pH.	25
5.17 Zegar cyfrowy.	25
5.18 Smartfon.	25
5.19 Telewizor wyświetlający ekran domyślny.	26
5.20 Telewizor wyświetlający ekran z podpowiedzią.	26
5.21 Układ okresowy pierwiastków.	26
5.22 Książka.	27
5.23 Waga.	27
5.24 Stół w szufladami i szafkami.	27
5.25 Drugi stół.	27
5.26 Zdjęcia chemików.	28
5.27 Drzwi.	28
5.28 Lampy.	28
5.29 Probówka z niebieską cieczą.	28
5.30 Kartka z napisanym związkiem chemicznym przyczepiona do tablicy korkowej.	29
5.31 Papier wskaźnikowy.	29
5.32 Papierki wskaźnikowe po zetknięciu z probówkami.	29
5.33 Śmietnik.	29
5.34 Śmieć.	30
5.35 Zlewka	30
5.36 Ściana.	30
5.37 Podłoga.	31
5.38 Sufit.	31
5.39 Użycie uproszczonego kolidera dla modelu wagi.	32
5.40 Zastosowanie publicznych pól w komponencie.	34
5.41 Wartości Constraints komponentu Rigidbody.	37
5.42 Użycie Gizmos w edytorze Unity.	38
6.1 Zakładka project settings.	46
6.2 Poruszanie się w jaskini MiniCAVE.	48
6.3 Otwieranie sejfu na MiniCAVE.	49

6.4	Odnaleziony telefon.	50
6.5	Próba rozwiązania zagadki z telefonem.	50
6.6	Próba ustawienia kodu do kłódki.	51
6.7	Odblokowana kłódka.	51
6.8	Zauważenie włącznika światła.	51
6.9	Odnalezienie drugiego włącznika.	52
6.10	Wyświetlany kod na zegarze.	52
6.11	Znalezienie pierwszej kartki.	52
6.12	Przykładanie kartek do tablicy korkowej.	53
6.13	Przenoszenie kartki na tablicę korkową.	53
6.14	Wszystkie kartki znajdują się na tablicy korkowej.	53
6.15	Wyświetlona podpowiedź.	54
6.16	Próba wpisania daty do sejfu.	54
6.17	Nieprawidłowy kod.	54
6.18	Prawidłowy kod.	55
6.19	Wnętrze sejfu.	55
6.20	Antidotum położone na wadze.	55
6.21	Odnalezienie jednej z fiolek.	56
6.22	Próba ustawiania fiolek w odpowiednie miejsce.	56
6.23	Dalsza próba ustawienia fiolek.	56
6.24	Poprawne ustawienie fiolek.	57
6.25	Porównywanie koloru na papierku z skalą pH.	57
6.26	Dalsza próba porównywania koloru.	57
6.27	Prawidłowe ustawienie papierków wskaźnikowych.	58
6.28	Próba odblokowywania reszty klódek.	58
6.29	Wszystkie kłódki zostały odblokowane.	58

WYKAZ TABEL

5.1 Materiały emisyjne	42
5.2 Materiały transparentne	42
5.3 Materiały zanikające	43
5.4 Materiały zwykłe	44