**Uniwersytet Jagielloński w Krakowie**

Wydział Fizyki, Astronomii i Informatyki Stosowanej

**Paweł Kołodziej**

Nr albumu: 1153732

**Tytuł pracy dyplomowej**

Praca magisterska  
na kierunku Informatyka Gier Komputerowych

Praca wykonana pod kierunkiem

Dr Anny Sochockiej

<Instytut/Zakład>

Kraków 2023

**Spis treści**

1. **Wstęp x**
   1. Kontekst……………………………………………………………………………x
   2. Zakres pracy…………………………………………………………...x
2. **Struktura pracy x**
   1. Cel pracy…………………………………………………………………………...x
3. **Budowa mózgu x**

3.1 Budowa mózgu…………………………………………………………………….x

1. **Gry poważne x** 
   1. Gry poważne……………………………………………………………………….x
2. **Opis implementacji x**
   1. Użyte narzędzia…………………………………………………………………….x
   2. Implementacja……………………………………………………………………...x
   3. Tutorial dla użytkownika…………………………………………………………..x
3. **Opis testów i badań x**
4. **Podsumowanie x**
5. **Bibliografia x**

Rozdział 1

**Wstęp**

* 1. **Kontekst**

Niniejsza praca magisterska dotyczy zagadnienia edukacyjnej gry poważnej, która ma na celu ułatwić użytkownikom proces poznawania i uczenia się tego jak działa ludzki mózg. Interaktywne metody zdobywania wiedzy, przy zastosowaniu właśnie gier poważnych, mają spory potencjał w prześcignięciu tradycyjnych form uczenia się opartych na tekście, pod względem efektywności przyswajania wiedzy. Gry poważne dają możliwość do nauki która jest bardziej interaktywna i angażująca więcej zmysłów. Osoba ucząca się ma możliwość większej interakcji z obiektem którego włąściwośći poznaje, przez co proces zapamiętywania jest prostszy.

Poznawanie możłiwośći ludzkiego mózgu jest bardzo ważne nie tylko pod względem edukacyjnym, ale także może pomóc w wielu dzedzinach życia każdego z nas. Zrozumienie tego jak w istocie działa ludzki móżg, jest w stanie lepiej zrozumieć pdstaowwe procesy któ®e nimi rządza, jest w stanie lepiej wykorzystywać jego możliwośći, zdolnośći poznawcze… Dodatkowo w przypadku osób młodszych, poznanie funcdamentów działania tego ważngo narządu (?) daje dobry fundament do przyszłego zainteresowania się takimi dziedzinami nauki jak psycholgia, medycyjna, kognitywityka czy nauki związane z technologią.

Praca magisterska została wykonanna w formie gry poważnej, i ma za zadanie wskazać jak gry poważne, oraz niektóre jej jej elementy sa w stanie ułatwić proces uczenai si ei poznawania nowych zagadnień. Gra została wykonanan w silniku Unity, atomiast modele mózgu zostały wykonane w programie do grafiki trójwymiarowej Blender.   
 Praca zostałą podzielona na rozdiały, które przybliżają strukturę pracy magisterskiej, szczegóły implementacji, uzasadnią wybór narzędzi oraz tematyki. Przybliżona zostanie również kwestia samych gier poważnych, zjawiska grywalizacji, teoretycznej strony budowy mózgu. Praca zwieńczona jest testem i badaniem na użytkownikachm dotyczącym efektywnośći uczenia issię za pomocą takiej gry. Na końcu znajduje się tez bibliografia.

**Wstęp**

* 1. **Zakres pracy**

Celem pracy było wykonanie gry poważnej, któ®a umożliwia użytkownikowi zapozanie się z działąniemm i budowa ludzkiego mózgu. Zaprezentwoanie mózgu w formie interaktywnego trówjymairowego modelu miało za zadanie lepiej zobrazować użytkownikowi strukturę i budowę mózgu, a quiz i test wiedzy połączony z rankingiem wsyztskich użytkowników miał na calu wprowadzenie podstawowych mechanizmó grwazlizacji do gry. Zakres gry prezentuje się w następujący sposób:

- w menu głównym, gracz ma do wyboru zapoznanie siez budową i funcjonowaniem ludzkiego mózgu, lub przejść do bardziej wewntrznej części i zapoznać się z budową neuronu i fundamentalnych mechanizmóstojacyh za funkcjonowaniem mózgu,

- gracz w kolejnych ekranach ma możliwość interkacji z trójwymiarowym modelem mózgu, ma możliwość obracania modelu, zwiększania zmniejdszania, a takżę wyboru poszczególnych elementów, i wyświetelenia odpowiedniej informacji dotycznącej wybranego fragmentu mózgu,

- gracz ma możliwość przystąpienia do testu wiedzy, któ®y ma na celu sprawdzenie i przetestowanie czy udało się użytkownikowi przybliżyć pewne zagadnienia doyczace mżógu,

- gracz ma możliwość podglądau rankingu wynikó innych graczy,

1. **Wstęp + omówienie tego czego dotyczy praca (część pisemna) + struktura części pracy**

d

1. **Wstęp + omówienie tego co jest w części praktycznej**

d

1. **Mózg teoria**

Mózg pełni niezbędną rolę w unkcjonowaniu każdego człowikeka. Składa się z rócnych struktur, któ®e są zroganizowane i połączone w całość, a między nimi występuje odpoweidni hierarchia. Opis anatomii ludzkiego móżg można zacząć od kory mózgowej – jest to najbardziej zewnetrzna warstwa mózgu, któ®a okrywa swoją powierzchnią półkule. Skłąda się z wielu neuronów i stanowi cześć istoty szarej układu erwowego. dwóch półku – lewej i prawej. Półkule są ze sobą połączone spoidłę wielki,m. Kazdappółkula pełni swoje charakterystyczne funkcje. Lewa poółkula odpowiada bardziej za logiczne myslenie,… prawa za kreatywne…. Następnie każda z półkul słąda się z różnych płatów: czołowego, skroniowego, ciemieniowego, potylicznego. Płąty są następnie podzielone na mniejsze obszary, wśród których każdy z nich odpowiada za inne potrzebne do funkcjonowania role.

Npisac cos o neuronach.

1. **Gry poważne teoria**

Gry poważne to rodzaj gier które mają na celu coś więcej niż czytą rozrywkę,. Ich zadaiem jest dosyarzenie edykacji lub pomoc w rowziwazywaniu problemów. Są to narzędzia łączće w sobie lementry gier z walorami educkaycjnymi, informacyjnymi lub szkoelniowymi. CHarakteryują się one kilkoma cechami:

Według Johana Huizingi, gra jest aktywnością, któ®a ma charakter wolny, odizolowany od codziennych obowiązkó, w której uczestnicy działają na podstawie ustalonych zasad w celu osiągnięci aokreślonego celu. Holender zwraca uwagę na to żę gra sama w sobie jest sytuacją mającą na celu dostarczenie rozrywki ale także budowanie więzi społecznych. W książce „Homo Ludens: o znaczeniu gry w kulturze” prezentuje wiele aspektów gdy, takich jak swoboda: ograniczony czas i przestrzeń, dobrowolne uczestnictwo. Huizinga w swojej definicji skupia się na tym, że jest to forma odrębnej aktywności, oderwanej od codziennej aktywnośći społecznej, w której uczestnicy godzą się na pewne zasady i rywalizują w określonych ramach. Gra ma charakter wolny i dobrowolny w przeciwieństwie do codziennych czynności wykonywanych przez każdego z nas.

David Michael, Sande Chen w książce “Serious Games: Games that Educate, Train, and Inform” podają definicję gry poważnej jako interaktywnej aplikacji, posiadającej zasady gry, która ma za zadanie dostarczenie wartości innej niż czysta rozrywka. Tą wartością możę być edukacja, szkolenie, lub kwestia informacyjna w formie któ®a jest bardziej przystępna dla użytkownika niż tradycyjna forma tesktsowa. Głónym celem takiej gry jest przekazanie określonych treśći użytkownikowi. Kluczowymi elementami gry poważnej są przede wsyztskm: mechanizmy standarowej gry, czyli posiadanie zasad, pewnego celu, rodzaju wyzwania i ma to na celu wzbudzenie w graczu zaangażowania w grę. Kolejnym elementem jest wspomniana wcześniej wartość edukacyjna lub informacyjna lub szkoleniowa, któ®a ma zostać w trakcie gry w pewien sposó zaspokojona.

W książce "Serious Games: Mechanisms and Effects" autorstwa Ute Ritterfeld, Michaela Cody'ego i Petera Vordere, możemy znaleźc podobą definicję gry poważnej, mówiącą o tym żę gra poważna to gra która ma na celu dostarczenie rozrywki jednocześnie stawiając przed graczem pewne zadania, któ®a mają na celu przekazać pewne określone „poważne” treści, czyli treści edukacyjne, informacyujne, szkoleniowe. Według autorów gra poważna powinna spełniać kolka krytweiów: aspekt rozrywkowy któ®y powinien pryciągać i zachęcać użytkowników do grania (ale w przeciwieńśtwei do tradycyjnych gier, gra poważna ma założony cel inny niż rozrywka). Gra poważna powinna wykorzystywać mechanizmy grytakie jak reguły, nagroda, wyzwanie, rywalizacja, interaktywność co pozwala wciągnąć gracza w naukę.

Możę tu coś dopisać z krótkiego paperu, któ®y mam w folderze z prezentacjami na semiarium

1. **Użyte narzędzia**
   1. Unity

Silnik Unity jest jednym z najbardziej rozpowszechnionych i uniwersalnych narzędzi do tworzenia gier. Napisać kto go tworzy, kiedy został stworzony itd. Szczególną popularność posiada w branży gier komputerowych, czego dowodem jest jego zastosowanie w eilu projektach komercyjnych. Jego znaczenie wynika z wszechstronnośći, ale też innowacyjnych funkcjoalnosći, wspraraaiw dla wielu pratorm, latwosci używania (do czego przyczynia się też ogromne wsparcie społćzeniscuzytkownikow, którzy tworzą dobrowolnie bardzo dużo treści o charakterze szkoleniowym, któ®e pomaają nowym użytkownikom tego silnika na łatwiejsze wejście do tej technologii, a bardziej zawansowanym użyknoikom oferują pomoc w rozwiązywaniu problemów). Wszechstornność silnika unity objawia się w tym ze można w nim tworzyć gry na różne platformy taki ejak komputery osobiste, smartwfony, konsole lub nawet gry w wirtualnej rzecywitoi lub rozszerzonej rzeczyiwstoci. Elastyczność tego silnika przejawia się również w mozłoiwośći importowania własnych assetów lub wtyczek, lub wykorzystywaia tego typu materiałów któ®e są publiczne deostępne na zewnatrznych stronach internetowych.

Tak jak wspomniano wcześniej, łątwośc użytkowania silnika unity wynika nie tylko z intuicyjnego interfejsu, ale przede wsyztskim bardzo bohatej dokumentacji i wsparciu społeczności użytkowników. Wsparcie społeczności przejawia się w ogromnej liczbie materiałów na forach, filmó na platformach wideo, na któ®ych można zdobyć wiedzę bez względu na to na jakim poziomie zaawansowania się jest. Poczatkującym użytkownikom zostaje obniżony próg wejścia do silnika przez wzgląda na bardzo duż ą liczbe filmo instruktazowych przybliżających podstaowe zagadnienia, natomiast bardziej doswiadzceni użytkownicy, tworzący zaawansowane proejkty, jeżeli natrafią na jakiś porblem związany z silnikiem, mogą go bardzo szybko i skutecznie rozwaiżac na forach. Jest to nieoceniona przeawga tego silnika, któ®a została wytworzona organicznie przez użytkowników, a któ®ej brak przyczyniłby się być może do duoz mniejszej popularności silnika, ale przede wszystkim do dużo kniejszego komfrtu użytkowania. Sam silnik może również być udoskalany przez konakt ze swoją społęcznośćia.

Silnik unity oferuje różne zaawansowane narzędzia i funkcje któ®e sprawiają z e tworzenie gier jest prstrzze. Dotyczy to podstawowych zagadnie jak i bardziej skompikowanych procesów.

Silnik ten posiada również pewne wady takie jka na przykład wydajność. Silnik Unity jest wymagający jeśli chodzi o zasoby komputera, szczególnie w przypadku bardziej skompikowanych gier. Możeto proawdzic do koneicznosci optymializacji grylub zwiększenia zasobów sprzętowych. Kolejną wadą jest dość spora liczba oraz rozmiar generowanych plików. Kolejną wadą jest to ze przej???

5.1.2 Inne narzędzia

Unreal Engine – jest to również bardzo popularny silnik, na któ®ym zostyało stworzonych wiele komenrycjnych projektów i gier. Rozwiany jest przez Epic games. Podobnie jak unity oferuje bardzo dużą palete zaawansowanych narzęzdi, któ®e pozwalają na tworzenie projektów o całym spektrum zaawansowania. Możliwe jest tworzenie scen, grafiki 3d, systemów fizyki … dopisać cos.

Godot engine – napisac cos

1. **Opis implementacji**
   1. Zarys ogólny

A screenshot of a computer

Description automatically generated with low confidence

***InputHandler.cs***

S

***Menu.cs***

Jest to skrypt służący do przemieszczania się między kolejnymi scenami w grze. Zawiera on proste metody, zmieniające obecną scenę.

**public** **void** nextScreen()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

Metoda *nextScreen()* przełącza obecną scenę, na kolejną za pomocą argumentu *SceneManager.GetActiveScene().buildIndex + 1,* natomiastmetoda ta przyjmuje jako argument liczbę całkowitą, dzięki czemu możliwe jest przełączenie się do konkretnej sceny.

***ZoomMouse.cs***

Jest to skrypt, służący do przybliżania i oddalania modelu 3D za pomocą myszy. Zmienne zoomSpeed, maxZoomOut, maxZoomIn służą do kontrolowania prostych parametrów związanych z prędkością przybliżania, maksymalnego przybliżenia oraz oddalenia modelu od kamery w scenie.

**public** **class** ZoomMouseWheel : MonoBehaviour

{

**private** **float** zoomSpeed = 0.5f;

**private** **float** maxZoomOut = 0.5f;

**private** **float** maxZoomIn = 1.5f;

**private** **float** zoom;

**void** Update()

{

zoom = Input.GetAxis("Mouse ScrollWheel");

transform.localScale += **new** Vector3(zoom, zoom, zoom) \* zoomSpeed;

**if**(transform.localScale.x < maxZoomOut)

{

transform.localScale = **new** Vector3(maxZoomOut, maxZoomOut, maxZoomOut);

}

**else** **if** (transform.localScale.x > maxZoomIn)

{

transform.localScale = **new** Vector3(maxZoomIn, maxZoomIn, maxZoomIn);

}

}

}

W metodzie Update(), znajdują się dwie instrukcje warunkowe *if()*, których zadaniem jest uniemożliwienie oddalenia lub przybliżenia powyżej zadeklarowanej wartości zmiennej *maxZoomOut* i *maxZoomIn*.

***MouseHover.cs***

Jest to skrypt, który pozwala użytkownikowi na kolorowe podświetlenie wybranego obszaru modelu 3D, za pomocą najechania na ten obszar kursorem myszy.

**public** **class** MouseHover : MonoBehaviour

{

**private** Color newSphereColor;

**private** GameObject sphere;

**private** Renderer sphereRenderer;

**void** Start()

{

sphereRenderer = sphere.GetComponent<Renderer>();

}

**private** **void** OnMouseOver()

{

**if** (GrabRotation.isClickingTurnedOn)

{

activateColor();

}

}

**private** **void** OnMouseExit()

{

**if** (GrabRotation.isClickingTurnedOn)

{

defaultColor();

}

}

**public** **void** activateColor()

{

sphereRenderer.material.SetColor("\_Color", newSphereColor);

}

**public** **void** defaultColor()

{

newSphereColor = **new** Color(1.0f, 1.0f, 1.0f, 0.0f);

sphereRenderer.material.SetColor("\_Color", newSphereColor);

}

}

Metoda *OnMouseOver()* służy do obsługi zdarzenia w którym użytkownik „najechał” kursorem myszy na wybrany fragment modelu. Znajdująca się wewnątrz instrukcja warunkowa *if()* ma za zadanie manipulować podświetlaniem, i wyłączać tą opcję za pomocą zmiennej *GrabRotation.isClickingTurnedOn*. Zmienna ta jest statyczna i jej wartość może być zmieniona w innym skrypcie. Metoda *OnMouseExit()* działa bardzo analogicznie, natomiast obsługuje zdarzenie w którym kursor myszy opuścił podświetlany obszar.

Metoda *activateColor()* ma za zadanie zmienić kolor wybranego fragmentu modelu, tak aby był on wizualnie wyróżniony. Metoda *defaultColor()*, służy do przywrócenia oryginalnego koloru fragmentowi modelu. Jest tu mowa o fragmencie modelu, ponieważ intencją tego skryptu było podświetlenie wybranego fragmentu, a nie całego modelu (czyli na przykład podświetlenie jednego płata mózgu).

***BrainRotation.cs***

Jest to skrypt, którego zadaniem jest obracanie całego modelu mózgu.

**public** **class** BrainRotation : MonoBehaviour

{

**public** **float** rotationSpeed = 1000f;

**public** **bool** isBrainRotating = **false**;

**public** Transform centerOfRotation;

**void** Update()

{

**if** (isBrainRotating)

{

**float** mouseX = Input.GetAxis("Mouse X");

**float** mouseY = Input.GetAxis("Mouse Y");

transform.RotateAround(centerOfRotation.position, Vector3.down, mouseX \* rotationSpeed \* Time.deltaTime);

transform.RotateAround(centerOfRotation.position, Vector3.right, mouseY \* rotationSpeed \* Time.deltaTime);

}

isBrainRotating = **false**;

}

}

Zmienna *isBrainRotating* jest flagą, której ustawienie na wartość *true* w skrypcie *GrabRotation.cs*, przekazuje informację o tym, że model może się obracać. Jest ona istotna z tego względu, że jej brak powodowałby obracanie się modelu cały czas, zgodnie z ruchem myszy.

***GrabRotation.cs***

Jest to skrypt, którego zadaniem jest obsługa zdarzenia, gdzie użytkownik przytrzymuje lewy przycisk myszy na danym fragmencie modelu. Szczególnie ważne jest tutaj rozróżnienie sytuacji w której nad danym fragmentem modelu następuje naciśnięcie myszy, a naciśnięcie i przytrzymanie. W przypadku „naciśnięcia i przytrzymania” użytkownik może obracać model, a w przypadku zwykłego „kliknięcia” następuje przejście do trybu tekstowego, w którym użytkownik może przeczytać informacje tekstowe na temat wybranego fragmentu.

**public** **class** GrabRotation : MonoBehaviour

{

**public** BrainRotation brainRotation;

**public** MoveAnimation moveAnimation;

**private** **int** isDragCounter = 0;

**private** **const** **int** dragTrigger = 15;

**public** **bool** leftSideInactiveMode = **false**;

**public** **static** **bool** isClickingTurnedOn = **true**;

**private** **void** Start()

{

isClickingTurnedOn = **true**;

leftSideInactiveMode = **false**;

}

**void** OnMouseDrag()

{

**if** (isClickingTurnedOn)

{

isDragCounter++;

**if** (isDragCounter > dragTrigger)

{

brainRotation.isBrainRotating = **true**;

}

}

}

**private** **void** OnMouseUp()

{

**if** (isClickingTurnedOn)

{

**if** (isDragCounter <= dragTrigger)

{

moveAnimation.enableSideLeftAnimation();

}

isDragCounter = 0;

}

}

}

W skrypcie używane są metody *OnMouseDrag()* oraz *OnMouseUp()*, których zadaniem jest obsługa naciśnięcia przycisku myszy nad danym obszarem oraz zwolnienie tego przycisku.

W metodzie *OnMouseDrag()*, znajduje się mechanizm rozróżnienia naciśnięcia przycisku myszy, oraz naciśnięcia i przytrzymania. Możliwe jest to dzięki licznikowi *isDragCounter*, który rozpoczyna inkrementację w momencie naciśnięcia myszy. Jeżeli użytkownik przestał naciskać przycisk przed osiągnięciem pewnego *dragTrigger* (który jest stałą zmienną typu *int*), to uznawane jest to za „kliknięcie” i flaga *isBrainRotating* pozostaje w stanie false. W przeciwnym przypadku, gdy wartość *isDragCounter* przekroczyła progową wartość *dragTrigger*, uznawane jest to za „naciśnięcie i przytrzymanie” a wartość flagi *isBrainRotating* jest zmieniana na *true*, co jest istotną informacją dla skryptu *BrainRotating.cs.*

Metoda *OnMouseUp()* uruchamia się w momencie puszczenia przycisku myszy. Sprawdzane wtedy jest za pomocą instrukcji warunkowej *if()* czy użytkownik „kliknął” czy „kliknął i przytrzymał” fragment modelu. W przypadku pierwszej sytuacji, uruchamiana jest metoda *moveAnimation.enableSideLeftAnimation()* której zadaniem jest przełączenie sceny do trybu wyświetlania tekstu.

***MoveAnimation.cs***

Jest to skrypt, którego zadaniem jest przeniesienie sceny z trybu widoku, do trybu tekstowego. Jest uruchamiany w momencie gdy użytkownik przyciskiem myszy naciśnie na wybrany fragment modelu. Tryb tekstowy charakteryzuje się tym, że wybrany fragment mózgu zostaje podświetlony, cały model mózgu jest nieaktywny (czyli niemożliwe jest jego obracanie i przybliżanie) a obok zostaje zaprezentowany opis dotyczący tego konkretnego fragmentu mózgu. Po przeczytaniu tekstu, użytkownik ma możliwość wrócenia do trybu widoku (służy do tego przycisk „wyśrodkuj”) lub przejścia do kolejnej sceny (służy do tego przycisk „wejdź głębiej”).

**public** **class** MoveAnimation : MonoBehaviour

{

**public** **bool** isSideLeftAnimationOn = **false**;

**public** Transform brainPositionForDisplayingText;

**public** Transform cameraCentralPoint;

**public** Transform brainContainer;

**private** **float** movementSpeed = 6.5f;

**private** Vector3 vectorCentreOfLobeToCentreOfBrain;

**public** BrainDescriptionText brainDescriptionText;

**public** MouseHover mouseHover;

**public** GameObject centralizeBrainButton;

**public** GameObject goInsideButton;

Vector3 brainScaleSave;

**private** **void** Start()

{

brainScaleSave = brainContainer.transform.localScale;

setTextVisibility(**false**);

}

**void** Update()

{

vectorCentreOfLobeToCentreOfBrain = brainContainer.transform.position - transform.position;

**if** (isSideLeftAnimationOn)

{

moveBrainLeft();

}

**if** (BrainCentralization.isCentralizationMoveOn)

{

centralizationAdditionalActions();

GrabRotation.isClickingTurnedOn = **true**;

}

}

**public** **void** setTextVisibility(**bool** isTextVisible)

{

**if** (isTextVisible)

{

brainDescriptionText.textContainer.gameObject.SetActive(**true**);

centralizeBrainButton.gameObject.SetActive(**true**);

goInsideButton.gameObject.SetActive(**true**);

}

**else**

{

brainDescriptionText.textContainer.gameObject.SetActive(**false**);

centralizeBrainButton.gameObject.SetActive(**false**);

goInsideButton.gameObject.SetActive(**false**);

}

}

**void** moveBrainLeft()

{

brainContainer.transform.position = Vector3.Lerp(

brainContainer.transform.position,

brainPositionForDisplayingText.transform.position + vectorCentreOfLobeToCentreOfBrain,

Time.deltaTime \* movementSpeed);

**if** ((brainPositionForDisplayingText.transform.position.x + vectorCentreOfLobeToCentreOfBrain.x)

- brainContainer.transform.position.x < 0.3f)

{

mouseHover.activateColor();

setTextVisibility(**true**);

scaleUpBrain();

GrabRotation.isClickingTurnedOn = **false**;

}

}

**public** **void** enableSideLeftAnimation()

{

isSideLeftAnimationOn = **true**;

}

**public** **void** disableSideLeftAnimation()

{

isSideLeftAnimationOn = **false**;

setTextVisibility(**false**);

}

**void** centralizationAdditionalActions()

{

disableSideLeftAnimation();

scaleDownBrain();

mouseHover.defaultColor();

}

**void** scaleUpBrain()

{

brainContainer.transform.localScale = Vector3.Lerp(brainContainer.transform.localScale, brainContainer.transform.localScale \* 5.0f, 0.2f \* Time.deltaTime);

}

**void** scaleDownBrain()

{

brainContainer.transform.localScale = brainScaleSave;

}

}

Skrypt w metodzie *Start()* ustawia początkowe wartości, a metodą *Update()* aktualizuje wydarzenia na scenie. Zadaniem zmiennej *vectorCentreOfLobeToCentreOfBrain* jest przechwycenie pozycji odpowiednich punktów na scenie, dzięki czemu możliwa jest sytuacja w której po przejściu do trybu tekstowego, model przybliżony mózgu zostaje wyśrodkowany w stosunku do punktu znajdującego się w wybranym przez użytkownika fragmencie mózgu.

Metoda *setTextVisibility()* ma za zadanie wyświetlić boczny tekst, oraz odpowiednie przyciski.

Metoda *moveBrainLeft()* ma za zadanie przenieść w odpowiednim czasie cały model mózgu z punktu określonego zmienną *brainContainer.transform.position* do punktu *brainPositionForDisplayingText.transform.position*, dodatkowo przesuniętego o *vectorCentreOfLobeToCentreOfBrain.* Przesunięciedokonuje się za pomocą metody *Vector3.Lerp,* która wykonuje liniowąinterpolację między punktem początkowym a końcowym. Po przesunięciu następuje instrukcja warunkowa *if()* sprawdzająca czy model mózgu przesunął się na odpowiednią odległość. Jeśli tak się stało, to wybrany fragment mózgu podświetlany jest na wyróżniający kolor, wyświetlany jest boczny tekst, model mózgu jest odpowiednio powiększany a możliwość „klikania” w obszary mózgu zostaje tymczasowo wyłączona.

Kolejną istotną metodą jest *centralizationAdditionalActions()*, która wykonuje się w momencie gdy użytkownik nacisnął przycisk „wyśrodkuj”. Zmniejsza ona odpowiednio model mózgu, przywraca domyślny kolor wybranego fragmentu mózgu oraz umożliwia powrót do trybu widoku.

***BrainCentralization.cs***

Jest to skrypt, którego zadaniem jest przełączenie sceny z trybu tekstowego do trybu widoku. Towarzyszy temu usunięcie ze sceny opisu fragmentu mózgu, odpowiednich przycisków oraz przesunięcie modelu mózgu z powrotem do punktu centralnego sceny.

**public** **class** BrainCentralization : MonoBehaviour

{

**public** **static** **bool** isCentralizationMoveOn = **false**;

**public** Transform cameraCentralPoint;

**private** **float** movementSpeed = 6.5f;

**private** **const** **float** distanceBrainToCentreTriggeringFlag = 0.3f;

**void** Update()

{

**if** (isCentralizationMoveOn)

{

brainCentralizationMovement();

}

**if** (cameraCentralPoint.transform.position.x - transform.position.x < distanceBrainToCentreTriggeringFlag)

{

isCentralizationMoveOn = **false**;

}

}

**public** **void** tickCentralizationFLag()

{

isCentralizationMoveOn = **true**;

}

**public** **void** brainCentralizationMovement()

{

transform.position = Vector3.Lerp(

transform.position,

cameraCentralPoint.transform.position,

Time.deltaTime \* movementSpeed);

}

}

Metoda *brainCentralizationMovement()* do przesunięcia modelu mózgu na środek sceny używa opisywanej wcześniej metody *Vector3.Lerp*.

***TestBackend.cs***

Jest to skrypt odpowiedzialny za test wiedzy. Zaprezentowany LISTING? Został skrócony do najbardziej istotnych fragmentów.

**private** **void** Update()

{

scoreText.text = score.ToString();

timeScoreText.text = timeScore.ToString();

**switch** (currentQuestionNumber)

{

**case** 1:

**if** (!enterOnceInUpdate)

{

buttonNextQuestion.SetActive(**false**);

Timer.StartTimer();

updateTextElementOnScreen(question, "Który płat mózgowy nie istnieje?", 30);

answers.Add(**new** Answer(answer\_A\_button, "czołowy", **false**));

answers.Add(**new** Answer(answer\_B\_button, "polityczny", **true**));

answers.Add(**new** Answer(answer\_C\_button, "skroniowy", **false**));

answers.Add(**new** Answer(answer\_D\_button, "ciemieniowy", **false**));

updateTextElementOnScreen();

enterOnceInUpdate = **true**;

}

**break**;

.  
.  
.

**case** 12:

**if** (!enterOnceInUpdate)

{

buttonNextQuestion.SetActive(**false**);

Timer.StartTimer();

updateTextElementOnScreen(question, "Osoba uderzyła się mocno w tył głowy, i uszkodziła płat potyliczny. Jaki zmysł mógł ucierpieć?", 30);

answers.Add(**new** Answer(answer\_A\_button, "wzrok", **true**));

answers.Add(**new** Answer(answer\_B\_button, "słuch", **false**));

answers.Add(**new** Answer(answer\_C\_button, "mowa", **false**));

answers.Add(**new** Answer(answer\_D\_button, "węch", **false**));

updateTextElementOnScreen(**true**);

enterOnceInUpdate = **true**;

}

**break**;

}

}

**public** **void** answerCorrectnessHandler(Answer answer, **bool** isLastQuestion)

{

**if** (answer.isAnswerCorrect)

{

answer.answerButton.onClick.AddListener(() => correctAnswerHandler(answer, isLastQuestion));

}

**else**

{

answer.answerButton.onClick.AddListener(() => wrongAnswerHandler(answer, isLastQuestion));

}

}

**public** **void** removeButtonListener(Answer answer)

{

answer.answerButton.onClick.RemoveAllListeners();

}

**public** **void** correctAnswerHandler(Answer answer, **bool** isLastQuestion)

{

Timer.StopTimer();

timeScore = Timer.GetElapsedTimeInSeconds();

score += Timer.calculateTimeToPoints();

answer.answerButton.image.color = Color.green;

**foreach** (**var** answer2 **in** answers)

{

**if** (!answer2.isAnswerCorrect)

{

answer2.answerButton.interactable = **false**;

}

}

**if** (isLastQuestion)

{

buttonGoToResume.SetActive(**true**);

}

**else**

{

buttonNextQuestion.SetActive(**true**);

}

}

**public** **void** wrongAnswerHandler(Answer wrongAnswer, **bool** isLastQuestion)

{

**foreach** (**var** answer **in** answers)

{

answer.answerButton.interactable = **false**;

}

wrongAnswer.answerButton.image.color = Color.red;

wrongAnswer.answerButton.interactable = **true**;

**if** (isLastQuestion)

{

buttonGoToResume.SetActive(**true**);

}

**else**

{

buttonNextQuestion.SetActive(**true**);

}

}

**public** **void** resetEverything(Answer answer)

{

removeButtonListener(answer);

answer.answerButton.image.color = Color.white;

}

**public** **void** printTextElementOnScreen(Button answer, **string** text = "", **int** x = -426, **int** y = 39, **int** fontSize = 47)

{

Text textPrefab = answer.GetComponentInChildren<Text>();

Text newText = Instantiate(textPrefab, transform);

newText.text = text;

newText.color = Color.white;

newText.fontSize = fontSize;

RectTransform rectTransform = newText.GetComponent<RectTransform>();

rectTransform.anchoredPosition = **new** Vector2(x, y);

}

**public** **void** printTextElementOnScreen(Text answer, **string** text = "", **int** x = -426, **int** y = 39, **int** fontSize = 47)

{

Text newText = Instantiate(answer, transform);

newText.text = text;

newText.color = Color.white;

newText.fontSize = fontSize;

RectTransform rectTransform = newText.GetComponent<RectTransform>();

rectTransform.anchoredPosition = **new** Vector2(x, y);

}

**public** **void** updateTextElementOnScreen(**bool** isLastQuestion = **false**)

{

**foreach** (**var** answer **in** answers)

{

**if** (answer.isButtonActive)

{

answer.answerButton.interactable = **true**;

}

**else**

{

answer.answerButton.interactable = **false**;

}

answerCorrectnessHandler(answer, isLastQuestion);

answer.answerButton.GetComponentInChildren<Text>().text = answer.answerText;

}

}

**public** **void** updateTextElementOnScreen(Text answer, **string** text, **int** fontSize = 47, **bool** isButtonActive = **true**)

{

answer.GetComponentInChildren<Text>().text = text;

answer.GetComponentInChildren<Text>().fontSize = fontSize;

}

**public** **void** nextQuestion()

{

currentQuestionNumber++;

enterOnceInUpdate = **false**;

**foreach** (**var** answer **in** answers)

{

resetEverything(answer);

}

answers.RemoveRange(0, 4);

}

}

skrypt

A picture containing screenshot, text, operating system, font

Description automatically generated

Gra rozpoczyna się od pobrania od użytkownika imienia.

Imię pobierane jest za pomocą skryptu *InputHandler.cs*.

A screenshot of a computer screen

Description automatically generated with low confidence

Następnie gracz przechodzi do ekranu w którym może wybrać czy chce przejść do trybu widoku budowy mózgu lub budowy neuronu.

A picture containing screenshot, brain

Description automatically generated

W przypadku budowy mózgu, tryb widoku podzielony jest na 3 etapy:

- półkule mózgu,

- płaty mózgu,

- przekrój mózgu,

A green and pink brain

Description automatically generated with medium confidence

A green and pink brain

Description automatically generated with low confidence

W każdym z tych etapów, użytkownik może wykonać te same akcje, czyli: przybliżanie i oddalanie widoku modelu 3D mózgu, obracanie modelu, zazaczenie wybrango fragmentu i wyświetlenie opisu dotyczącego danego fragmentu (tryb tekstowy). Następnie użytkownik może wybrać, czy powraca do trybu widoku, czy chce przejść do kolejnego etapu. Wyjątkiem jest ostatni etap (widok przekroju mózgu), po którym gracz ma do wyboru, powrócić do ekranu budowy mózgu, przejść do budpwy neuronu, lub przejść do testu wiedzy.

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

6.2 Tryb modelu

6.2.1 Przybliżanie/oddalanie i obracanie

6.2.2 Zaznaczanie konkretnego obszaru

6.2.2 Tryb opisu

6.3 Test wiedzy

6.4 Tabela wyników

6.5 Zapisywanie danych do pliku

1. **Tutorial dla użytkownika**

d

1. **Podsumowanie**

d

1. **Bibliografia**

d