1. Опис програмного засобу

Опис програмного засобу

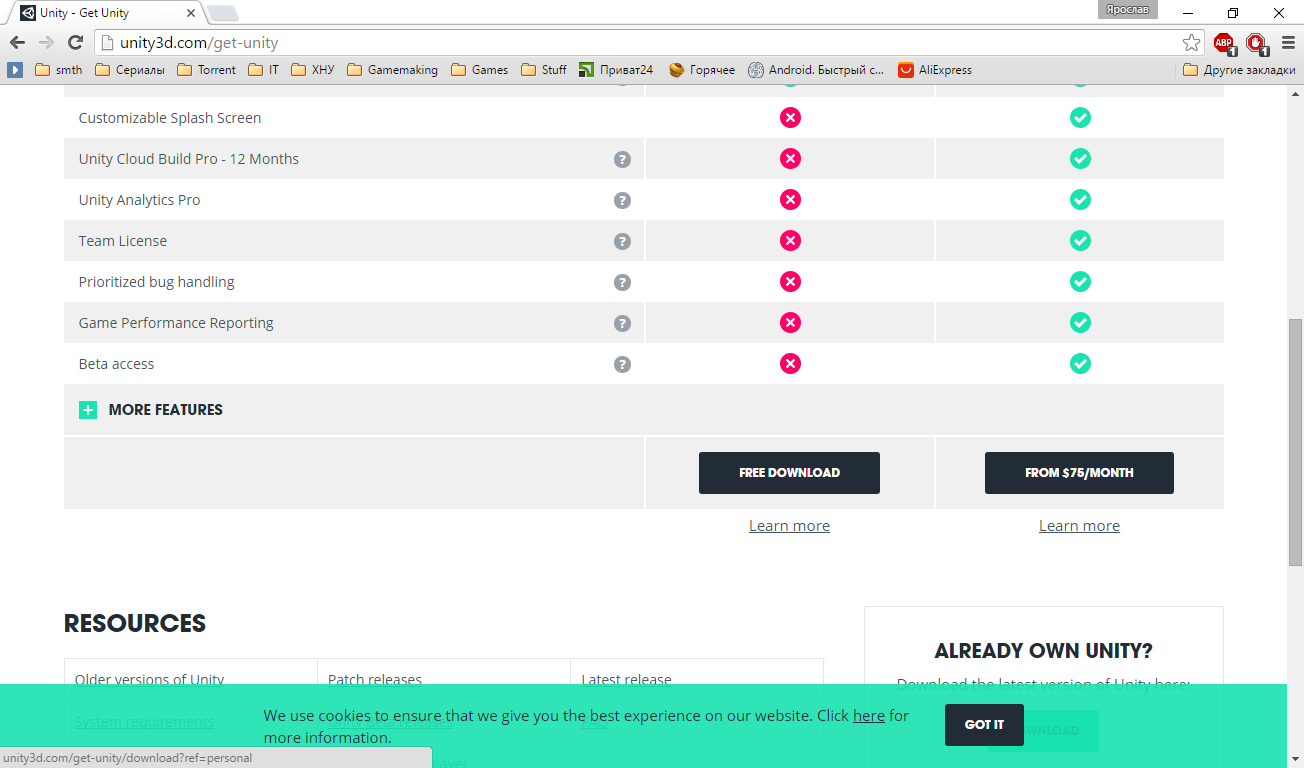
Unity - багатоплатформовий інструмент для розробки дво- та тривімирних застосунків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.

Є можливість створювати інтернет-застосунки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

(вікіпедія)

Отримати програмний засіб

Для того, щоб отримати Unity3D, потрібно перейти по посиланню http://unity3d.com/get-unity , після чого натискаємо кнопку “Free Download”.



Після підтвердження для завантаження веб-інсталятора та після його запуску ми проходимо шлях встановлення і завантаження повної версії програми, а також інтегрованого середовища розробки – Microsoft Visual Studio Community.

1. Старт проекту

Створення проекту гри

Після завершення завантаження та встановлення Unity3D вас зустріне вікно, в якому у вас запитуватимуть дані для аутентифікації.

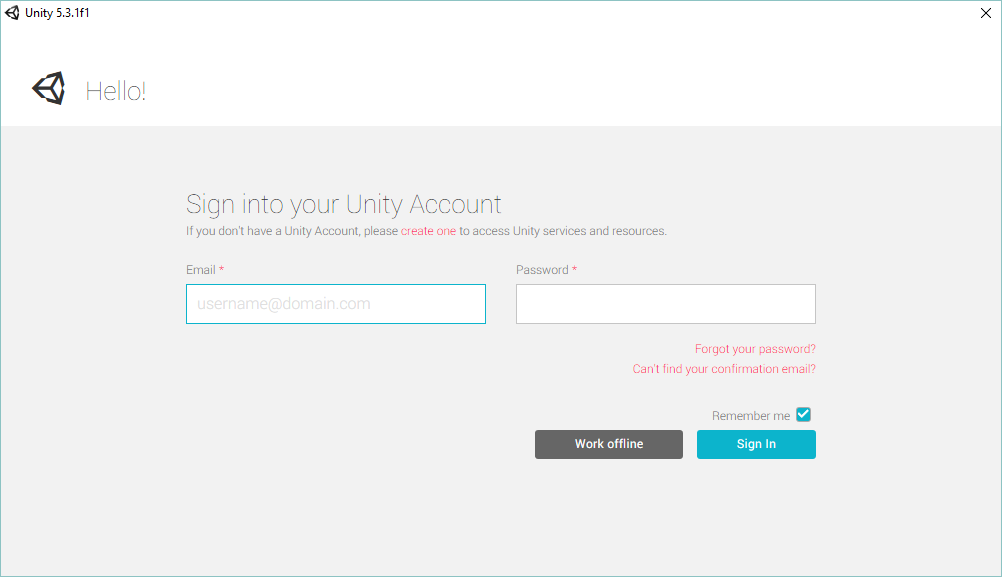


Рисунок – Форма автентифікації користувача

Є можливість працювати без автентифікації, але я рекомендую зареєструватися, оскільки є можливість того, що вам буде необхідно встановити перший готовий пакет рішень із Unity Asset Store.

Після проходження реєстрації, або якщо ви маєте логін та пароль, ми заповнюємо форму входу, далі ми за допомогою перемикачів вибираємо приватну ліцензію програми.

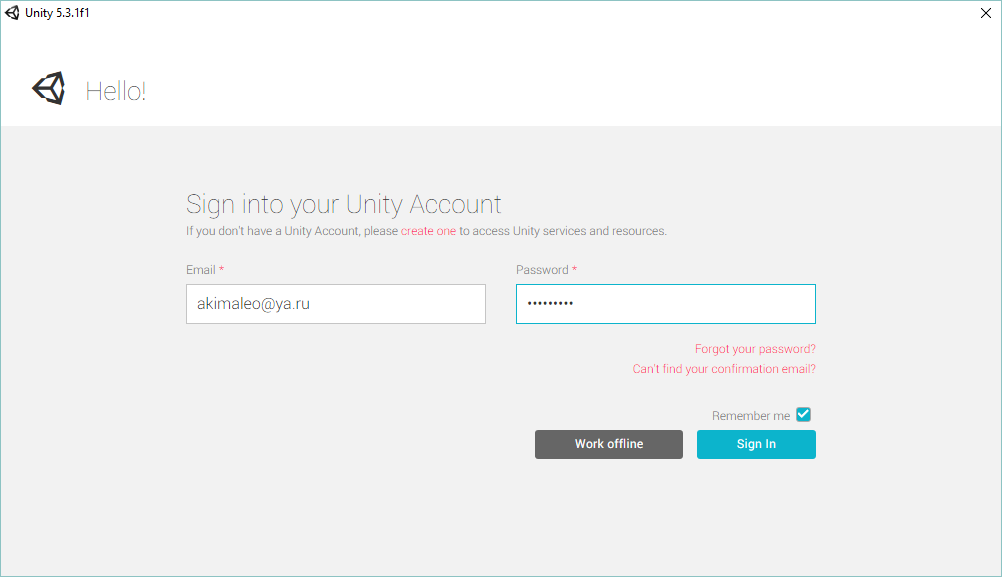


Рисунок – Заповнена форма автентифікації користувача

Після підтвердження даних для входу, бачимо вікно, в якому вам показуються всі ваші активні проекти, якщо це перший запуск програми, то поле для відображення активних проектів буде пустим.

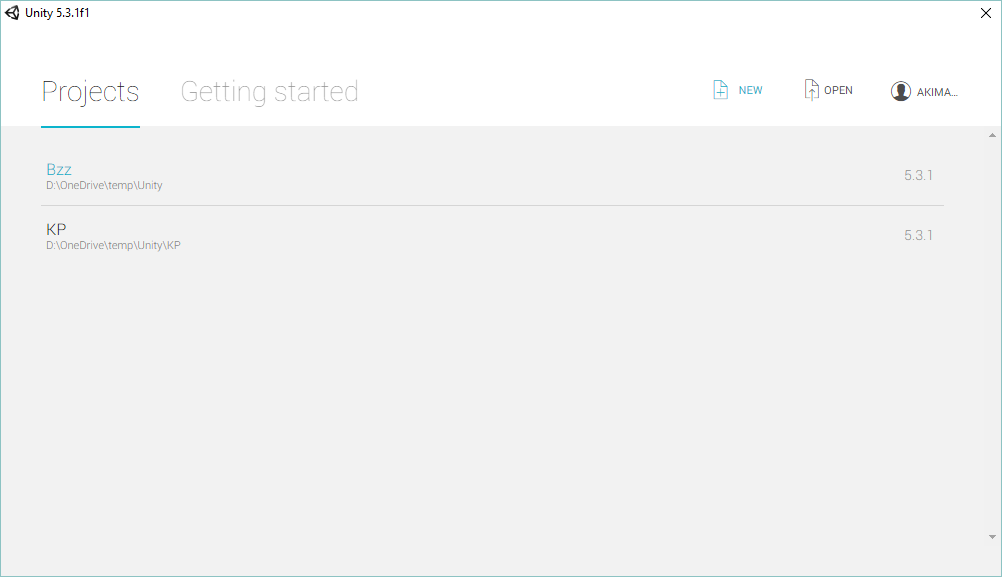


Рисунок – Вікно зі списком поточних проектів

Далі створимо власний проект, для цього натиснемо кнопку NEW і введемо назву нашої ігри, а також шлях, де будуть зберігатися вихідні файли проекту нашої гри.

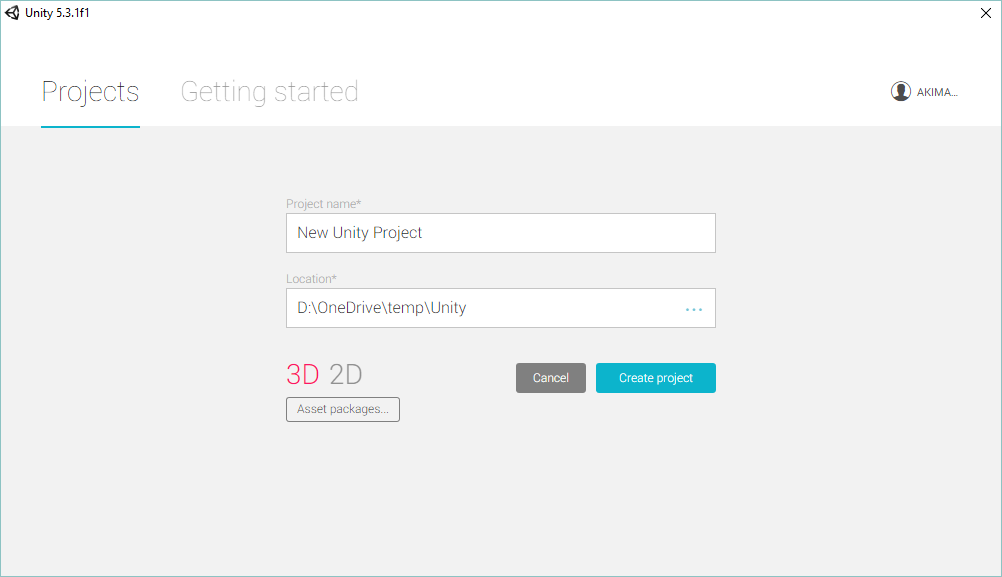
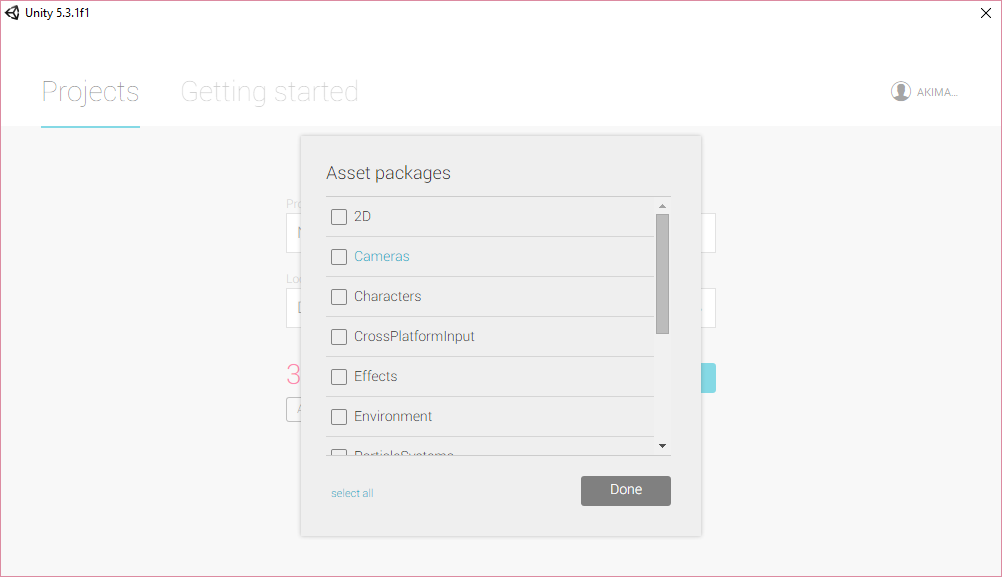


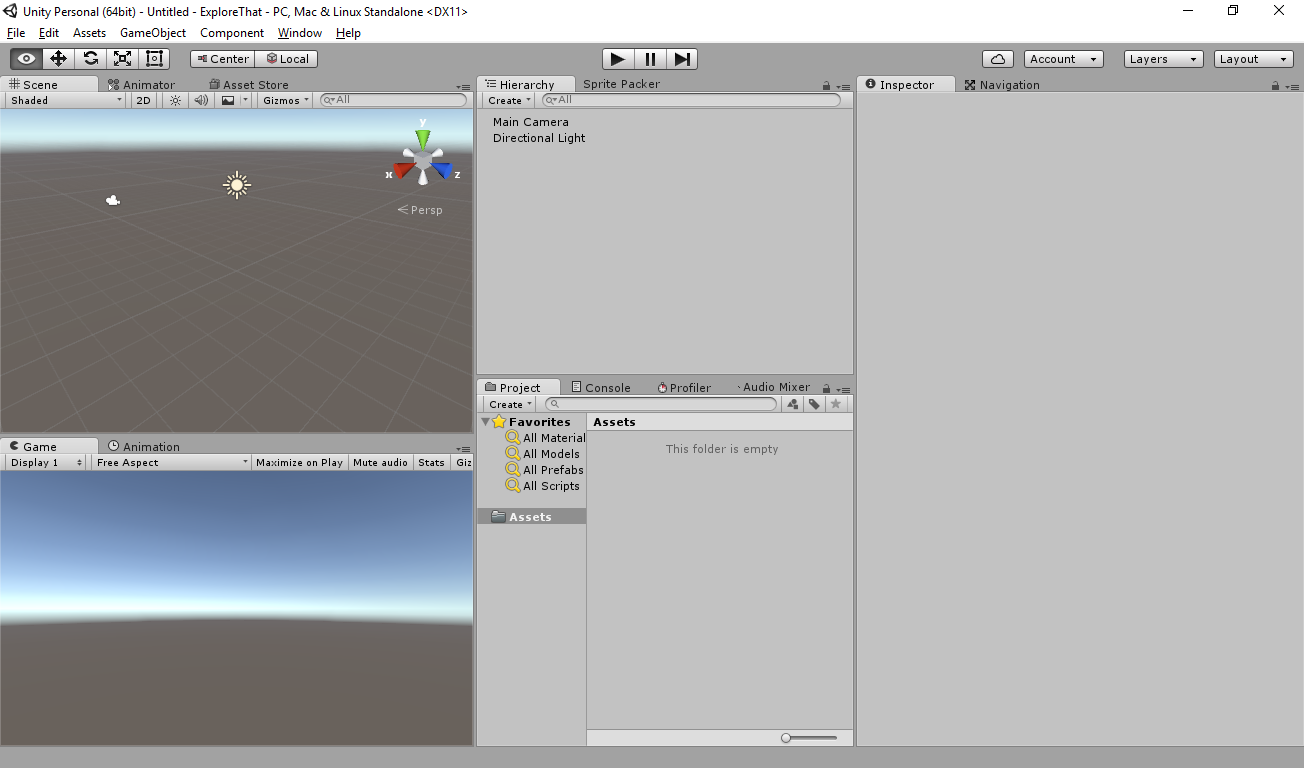
Рисунок – створення нового проекту

Також ми не будемо використовувати ніяких готових рішень в нашому проекті, тому, натискаємо кнопку “Asset packages..” і знімаємо всі позначки, якщо є.



Знайомство з редактором

Далі після підтвердження створення проекту нас зустріне пусте вікно редактора (рисунок ) .



Можливо, у вас розташування робочих областей буде іншим, це можна виправити просто перетягуючи вікна, так званий drag and drop інтерфейс.

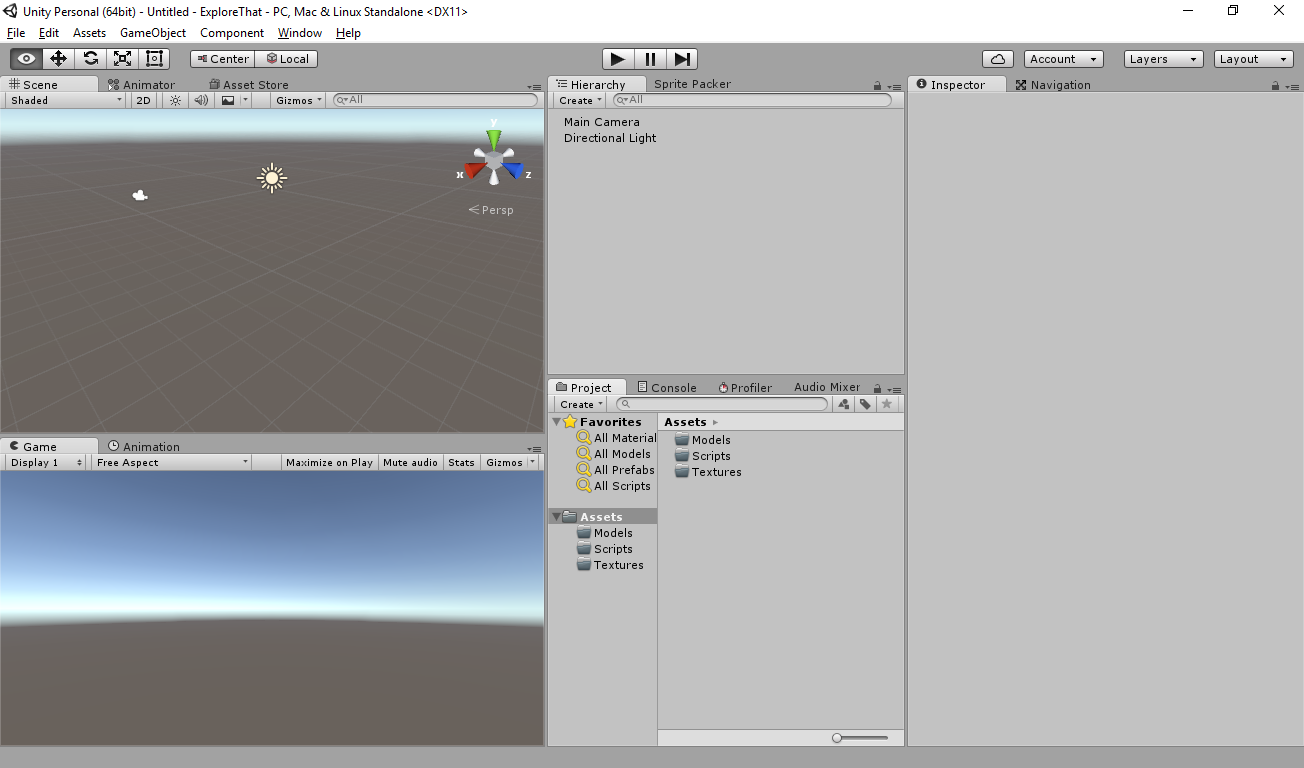
1. Розробка гри

Налаштування робочого процесу

Для того, щоб розпочати безпосередньо створення самої гри, ми створимо каталоги, в яких будемо зберігати файли гри. У вікні вкладки “Project” за допомогою контекстного меню і команд Create > Folder створимо папки Model, Scripts, Textures.



Після чого отримаємо вікно, в якому будуть відображені створенні нами користувацькі каталоги.



Створення ігрової сцени

Розпочнемо роботу зі створення ігрової сцени. При запуску програми, ігровий рушій Unity3D автоматично створив Main Camera і Directional Light.

Main Camera – ігровий об’єкт, який має компонент camera і застосовується для відображення ігрових об’єктів у грі. Варто розуміти різницю, у вкладці Scene, так званий «view port» ви оперуєте об’єктами на сцені, а у вкладці Game ви спостерігаєте ігровий процес безпосередньо як би це бачив гравець.

Directional Light – ігровий об’єкт глобального освітлення. За допомогою цього об’єкта освітлюється вся ігрова сцена, не залежно від позиції об’єкта, зміна освітлення буде відбуватися тільки під час повороту цього об’єкта, використовується як заміна сонцю.

Для створення нового ігрового об’єкту є два способи. Перший – це в стрічці меню відкрити випадаюче меню GameObject і вибрати необхідний об’єкт, після чого цей об’єкт буде відображатися у вкладках «Hierarhy» та «Scene».

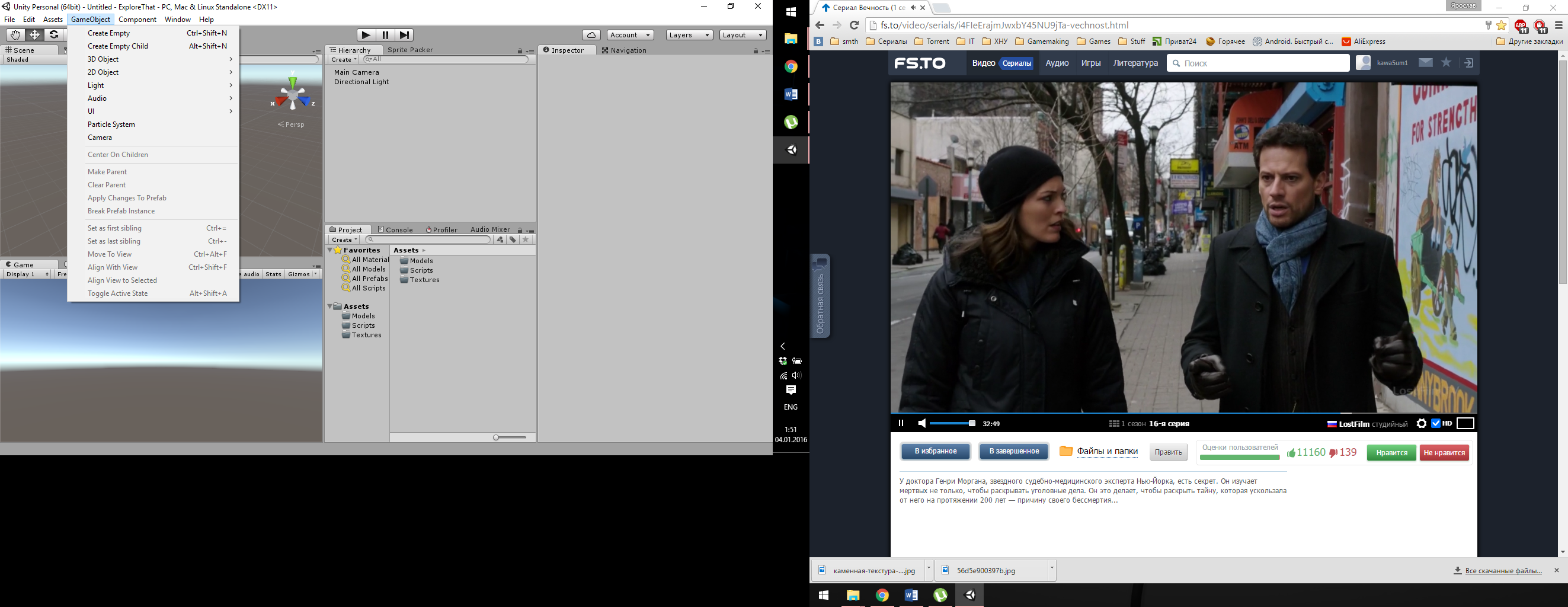


Рисунок –

Далі створимо персонажа, яким ми будемо керувати і за допомогою якого будемо взаємодіяти із ігровим простором.

/\*\*\*створення 3d об’єкта в cinema 4d

\*\*\* [силка](Models/Player/Player%20creation.docx)

\*\*\*/

Далі, для того, щоб добавити ігрову модель до нашого проекту – потрібно або перенести модель у папку з проектом, в каталог Assets/Models, або перенести файл моделі прямо у вкладку «Project».

Далі створимо пустий ігровий об’єкт «Create Empty», в який перемістимо камеру, а також модель нашого персонажа, розташуємо об’єкти так як показано на рисунку нижче. Є можливість того, що модель нашого персонажа буде великих розмірів, тому у вкладці «Inspector», в компоненті «Transform» змінну «Scale» підберіть так, щоб зберігати пропорції персонажа, в моєму випадку X, Y та Z я виставив на 0.01.

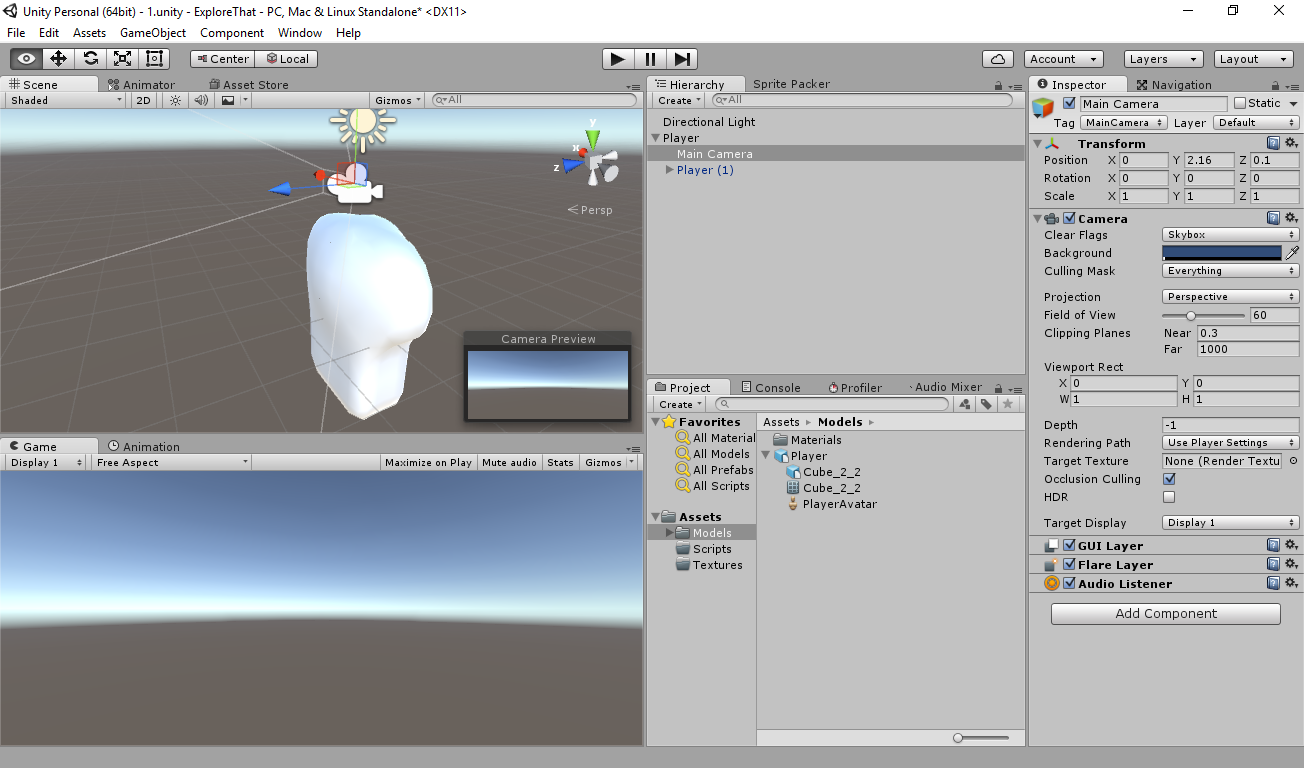


Рисунок -

Далі створимо ігровий об’єкт «Sphere» і поставимо його на місце голови, також варто підібрати відповідні розміри, аналогічно з моделлю персонажа. Таким чином отримаємо

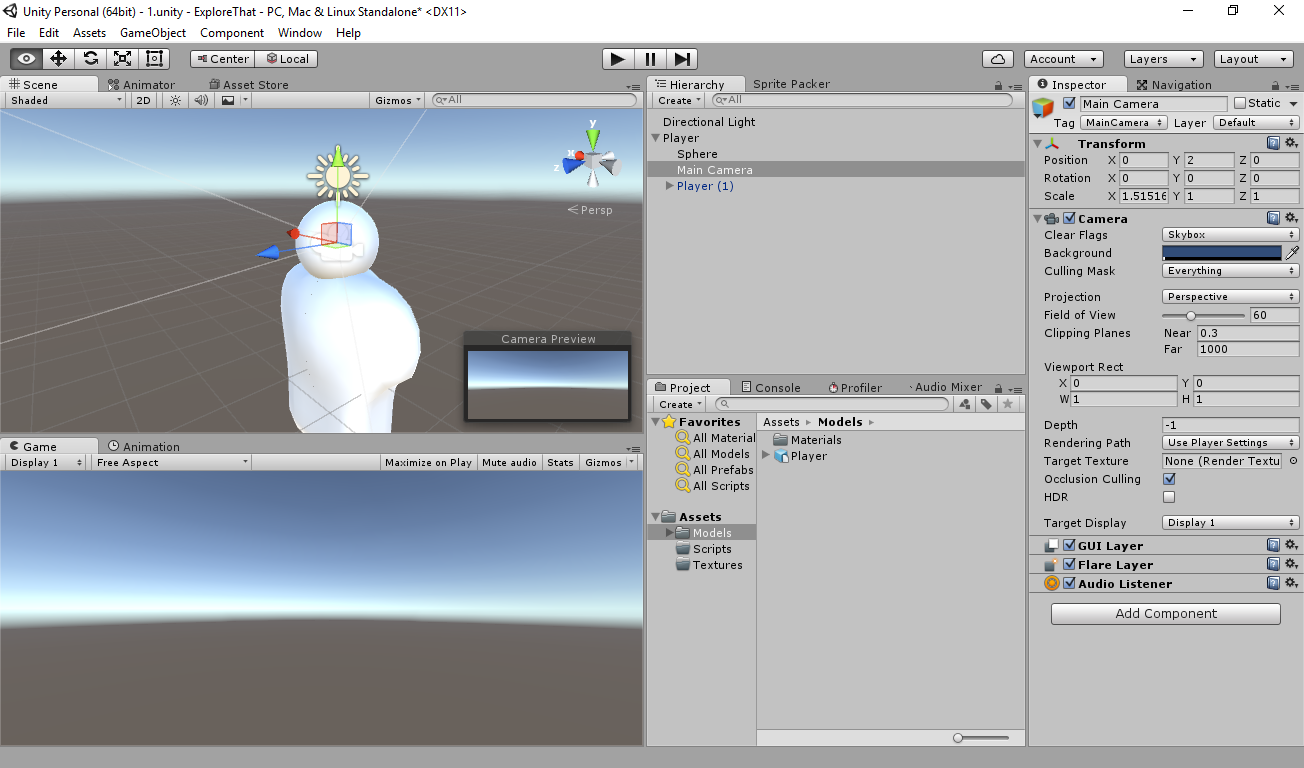


Рисунок -

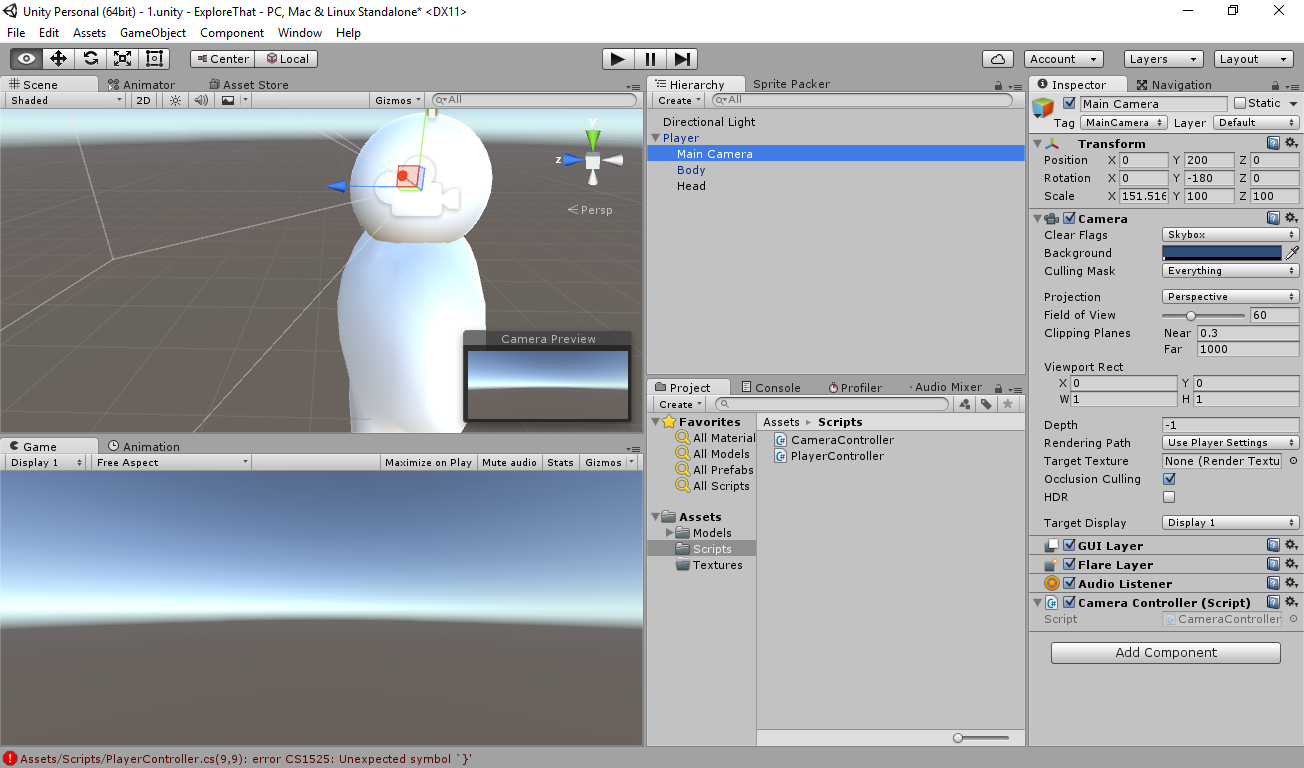


Рисунок –

Створення скрипту для керування камерою

Наслідуваний метод Start(){} використовується для ініціалізації об’єктів, змінних. Метод Update(){} використовується для здійснення операцій кожен кадр.

Далі розберемо створений клас для керування камерою.

Ми створюємо об’єкт класу Transform, цей клас використовується для того, щоб мати доступ до властивостей місцезнаходження GameObject. Після чого в ініціалізуючому методі Start() ініціалізуємо змінну player, для того, щоб мати доступ до головного об’єкту, який об’єднує нашого персонажа.

Реалізуємо можливість керування камери мишкою. Для цього оголосимо поза методами змінні типу float, які будуть використовуватися для повертання камери по осям x, y, а також для того, щоб мати змогу керувати чутливістю камери.

Перейдемо до методу Update(){}. В цьому методі ми получаємо інформацію про координати курсора миші, а також для перешкоджання повертанню камери вертикально на 360° ми використаємо метод Mathf.Clamp(), який застосовується для того, щоб утримувати значення в певних рамках, перше значення, яке передається у метод - зберігається у рамках, які задаються двома іншими параметрами.

Для реалізації функції повороту камери відносно осі x екземпляр класу Transform – player повернемо на те значення, яке ми отримали, відслідковуючи положення курсора по координаті x, для цього використаємо атрибут класу localEulerAngles, цей атрибут приймає значення класу Vector3, в який ми передаємо три значення для поворотів. Перше – це по осі x, яке ми отримали вище, далі зберігаємо кути поворотів тіла.

Далі збережемо поворот тіла по осі x, використовуючи метод Rotate() екземпляру класу Transform.

Лістинг класу CameraController представлений нижче:

public class CameraController : MonoBehaviour {

Transform player;

void Start () {

player = GameObject.Find("Player").GetComponent<Transform>();

}

float rotationX, rotationY = 0F;

public float sensitivity = 2F;

void Update () {

//Поворот камери і тіла мишою

rotationX = Input.GetAxis("Mouse X") \* sensitivity;

rotationY += Input.GetAxis("Mouse Y") \* sensitivity;

rotationY = Mathf.Clamp(rotationY, -60, 60);

transform.localEulerAngles = new Vector3(-rotationY,

transform.localEulerAngles.y,

transform.localEulerAngles.x);

//поворот тіла по y, замість камери

player.transform.Rotate(0, rotationX, 0);

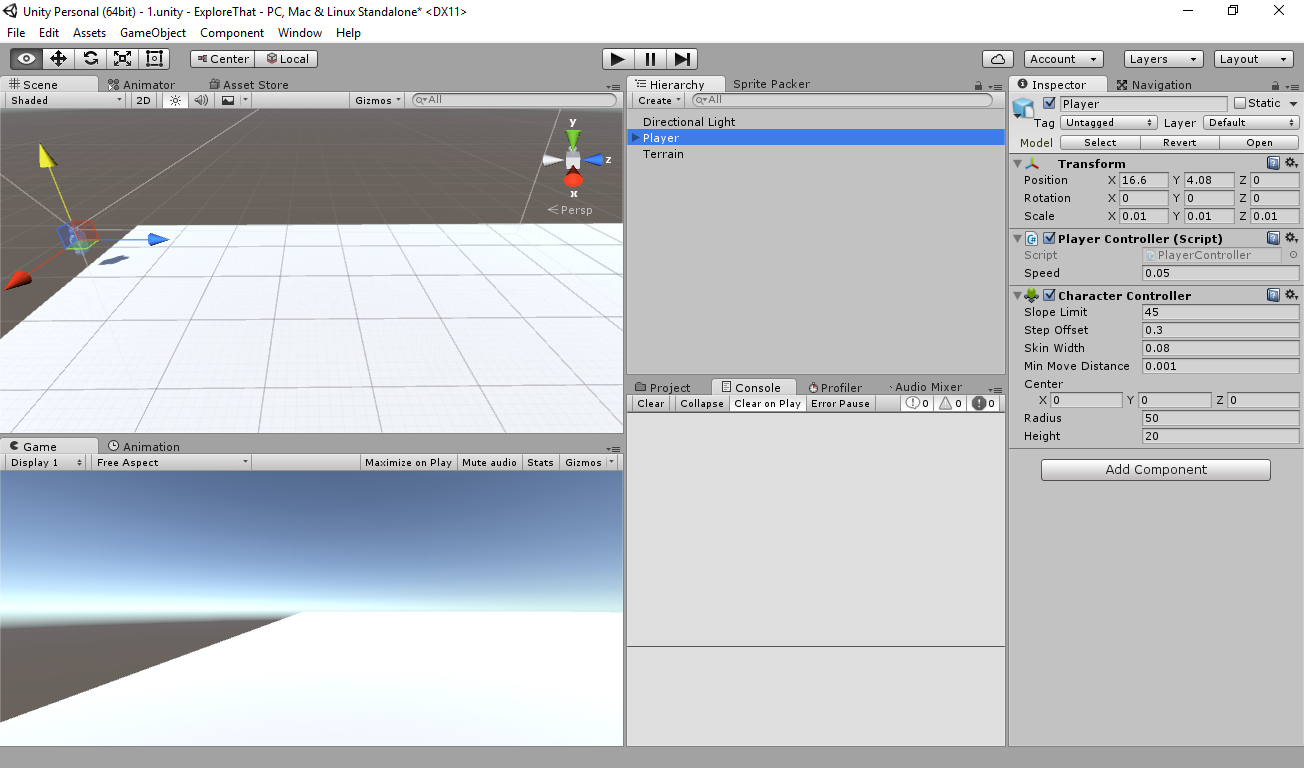
}

}

Створення скрипту для керування персонажем

Далі ми створимо клас для керування нашим об’єктом, надалі персонажем. Для того, щоб мати змогу керувати персонажем і взаємодіяти з об’єктами на ігровому полі для об’єкту нашого персонажу ми добавимо новий компонент CharacterController.

Варто відмітити, що кожен об'єкт сцени має певні компоненти, деякі з компонентів є обов'язкові для певних груп об'єктів, наприклад Transform для об’єктів ігрової сцени, а також кожен з цих компонентів має певні властивості і методи для роботи з цим об’єктом.



Для того, щоб отримати компонент для роботи, потрібно оголосити змінну типу цього компоненту, зазвичай назва класу, яка відповідає компонентові об’єкта співпадає. Наприклад, для того, щоб отримати компонент Character Controller потрібно оголосити змінну типу CharacterController chCtrl; після чого, бажано, у тілі методу ініціалізації Start(){} ініціалізувати оголошену зміну.

CharacterController – це компонент, який входить в стандартні компоненти ігрового двигуна Unity3D і має готові методи, за допомогою яких можна вирішувати певні поставленні задачі, в нашому випадку – це переміщення нашого персонажу. Для переміщення персонажу використовується метод Move(Vector3(x,y,z)).

Для присвоєння значення змінній використаємо метод GetComponent<T>();. Розглянемо дану функцію поближче. Метод GetComponent<T>(); застосовується для того, щоб мати змогу отримати певний компонент ігрового об’єкту. Цей метод має універсальний параметр <T>, де T – клас компоненту, який ми хочемо отримати. Існує також аналог цього методу, де замість універсального параметру <T> в параметри методу передається змінна типу string, яка містить в собі назву класу, наприклад запис:

(CharacterController)GetComponent("CharacterController")

Відповідає запису:

GetComponent<CharacterController>();

І присвоївши значення, які повертають ці методи змінній chCtrl, ми ініціалізували її, і тепер зможемо використовувати.

Для того, щоб реалізувати функцію ходьби ми будемо використовувати метод chCtrl.Move(Vector3). Де Vector3 – це вектор, який вказує напрям руху моделі персонажу. Для того, щоб визначити напрям руху ми використаємо тригонометричну властивість векторів і кутів. Наприклад ми маємо координати нашого персонажу, припустимо – це точка Р, і кут α, який показує, наскільки відхилене тіло. Завдяки цим даним ми зможемо розрахувати одиничний вектор, який показує напряму погляду персонажу. Для цього потрібно на значення Vector3(x,y,z); замінити. X = Cos(α) , Y = 0, так як ми не рухаємося по висоті за допомогою миші і Z = Sin(α).

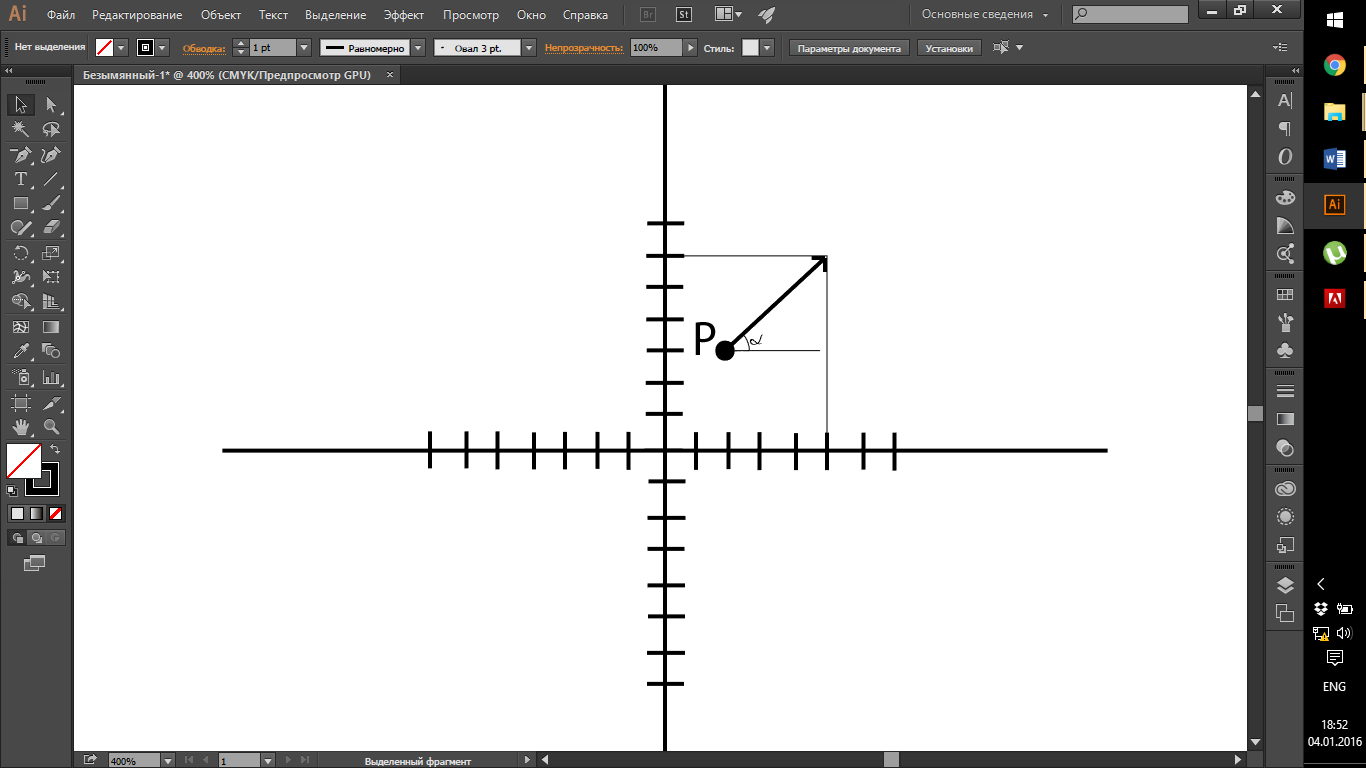


Рисунок -

Таким чином ми отримали таку лінію коду:

yDegrees = transform.rotation.eulerAngles.y \* Mathf.PI / 180;

forwardDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees)) \* speed;

Варто звернути увагу на те, що вектор ми множимо на змінну float speed, для того, щоб мати більшу гнучкість програмного коду, а також yDegrees типу float і forwardDirection типу Vector3 ми оголошуємо поза тілом методу для того, щоб не навантажувати збирач сміття. Далі потрібно реалізувати функціонал для переміщення персонажу, задля зменшення потреб обчислюваних ресурсів ми у змінну Vector3 moveDirection , будемо записувати всі зміни, які мають придатися у цю змінну. А коли вже всі змінні будуть вирахуванні, то тільки тоді ми за допомогою методу character.Move(moveDirection \* speed \* Time.deltaTime); задіємо всі зміни, також варто відмітити, що переміщення я домножую на значення статичної змінної класу Time.deltaTime, це зроблено для того, щоб персонаж рухався з однаковою швидкістю, не залежно від FPS.

Для відслідковування вхідного потоку з пристроїв введення, є статичний клас Input, а метод Input.GetKey(KeyCode.K), в який передається перерахування з клавіш на клавіатурі і при натисканні відповідної клавіші метод повертає булеве значення true. Для керування персонажем будемо використовувати стандартну розкладку “WASD”. Код набуде наступного вигляду:

if (Input.GetKey(KeyCode.W))

moveDirection += forwardDirection;

else if (Input.GetKey(KeyCode.S))

moveDirection += forwardDirection;

Також існують осі Vertical і Horizontal, за допомогою яких теж можна відслідковувати натискання клавіш. Для цього використаємо функцію Input.GetAxis("Vertical"). А також потрібно відслідковувати змінну значення осі, і при змінні значення ми змінюватимемо напрям руху, помножимо вектор на швидкість . Отримаємо код в такому вигляді:

if (Input.GetAxis("Vertical") != 0)

moveDirection += Input.GetAxis("Vertical")\* forwardDirection \* speed;

Для реалізації цієї задачі ми використовуватимемо другий метод. Тоді, для реалізації руху в сторони ми по аналогії використаємо тригонометричну властивість, тільки напрям погляду ми збільшимо на 90°. Таким чином ми матимемо вектор, що буде направлений перпендикулярно до погляду і паралельно до площини «землі». Аналогічно оголосимо значення sideDirection типу Vector3 поза методом, код матиме вигляд наступний вигляд:

yDegrees = (transform.rotation.eulerAngles.y + 90) \* Mathf.PI / 180;

sideDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees));

Далі потрібно реалізувати функцію переміщення персонажа в сторони, для цього добавимо такий рядок коду :

if (Input.GetAxis("Horizontal") != 0)

moveDirection += Input.GetAxis("Horizontal") \* sideDirection \* speed;

Отже, для простого керування персонажем матимемо клас Player Controller, який буде мати такий вигляд :

public class PlayerController : MonoBehaviour

{

CharacterController character;

void Start()

{

character = GetComponent<CharacterController>();//(CharacterController)GetComponent("CharacterController")

}

//movement speed

public float speed = 0.05f;

//forwardDirection that depends on body position

//moveDirection depends from forward direction and view angles

Vector3 forwardDirection, moveDirection, sideDirection;

private float yDegrees;

void Update()

{

moveDirection = Vector3.zero;

yDegrees = transform.rotation.eulerAngles.y \* Mathf.PI / 180;

forwardDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees));

yDegrees = (transform.rotation.eulerAngles.y + 90) \* Mathf.PI / 180;

sideDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees));

//WASD MOVEMENT

if (Input.GetAxis("Vertical") != 0)

moveDirection += Input.GetAxis("Vertical")\* forwardDirection \* speed;

if (Input.GetAxis("Horizontal") != 0)

moveDirection += Input.GetAxis("Horizontal") \* sideDirection \* speed;

character.Move(moveDirection\* Time.deltaTime);

}

}

Для збільшення функціоналу коду реалізуємо можливість присідання, бігу і стрибків. Почнемо із присідання: для цього область, яку займає Character Controller просто зменшимо і піднімемо на декілька одиниць вище, так як зменшення області колізії Character Controller стискає до центру, а камера, завдяки якій ми бачитимемо, залишатиметься набагато вище.

Зелена стартова область та значення, які характеризують область фізичної взаємодії з фізичними об’єктами на ігровій сцені, зображенні на рисунку



Рисунок - Змінні без модифікацій

Далі потрібно визначити які значення повині бути для того, щоб моделювати присідання, підібравши, матимемо значення зображенні на рисунку нижче.

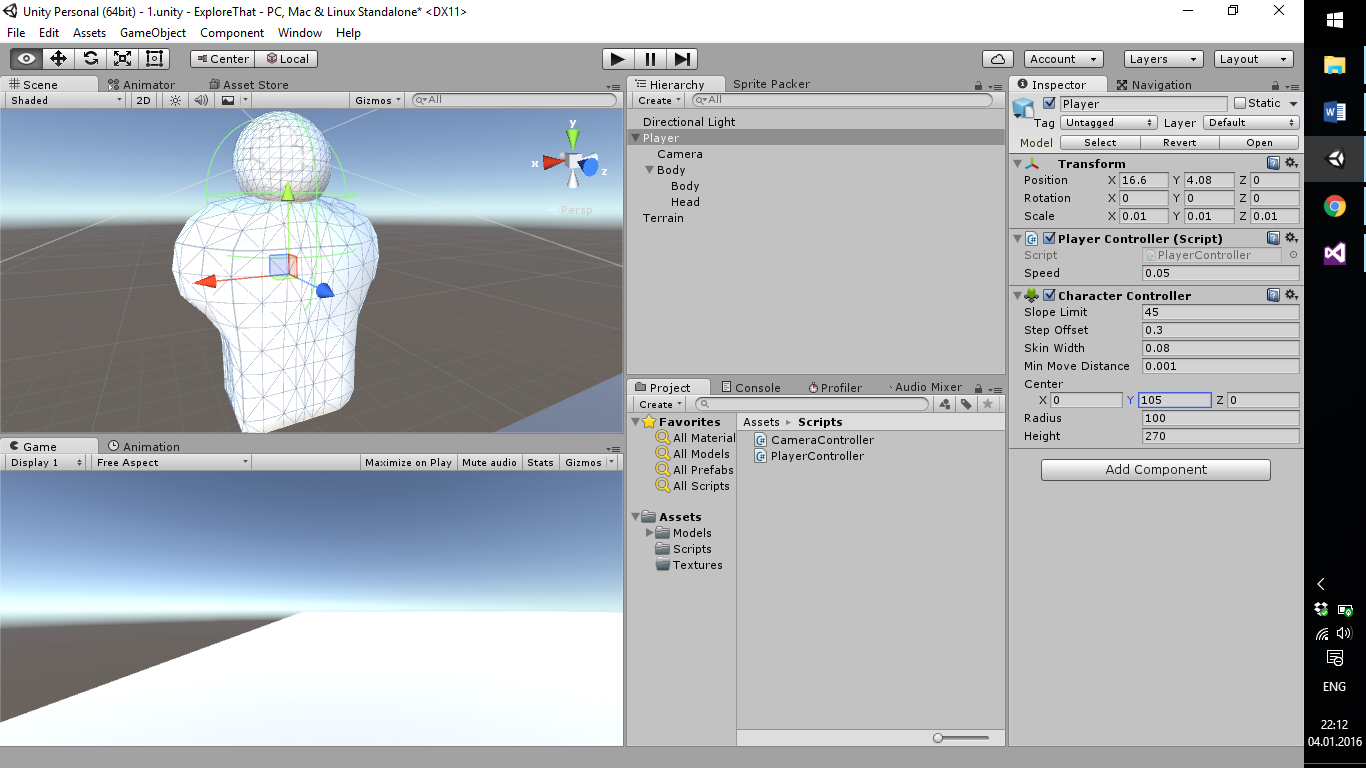
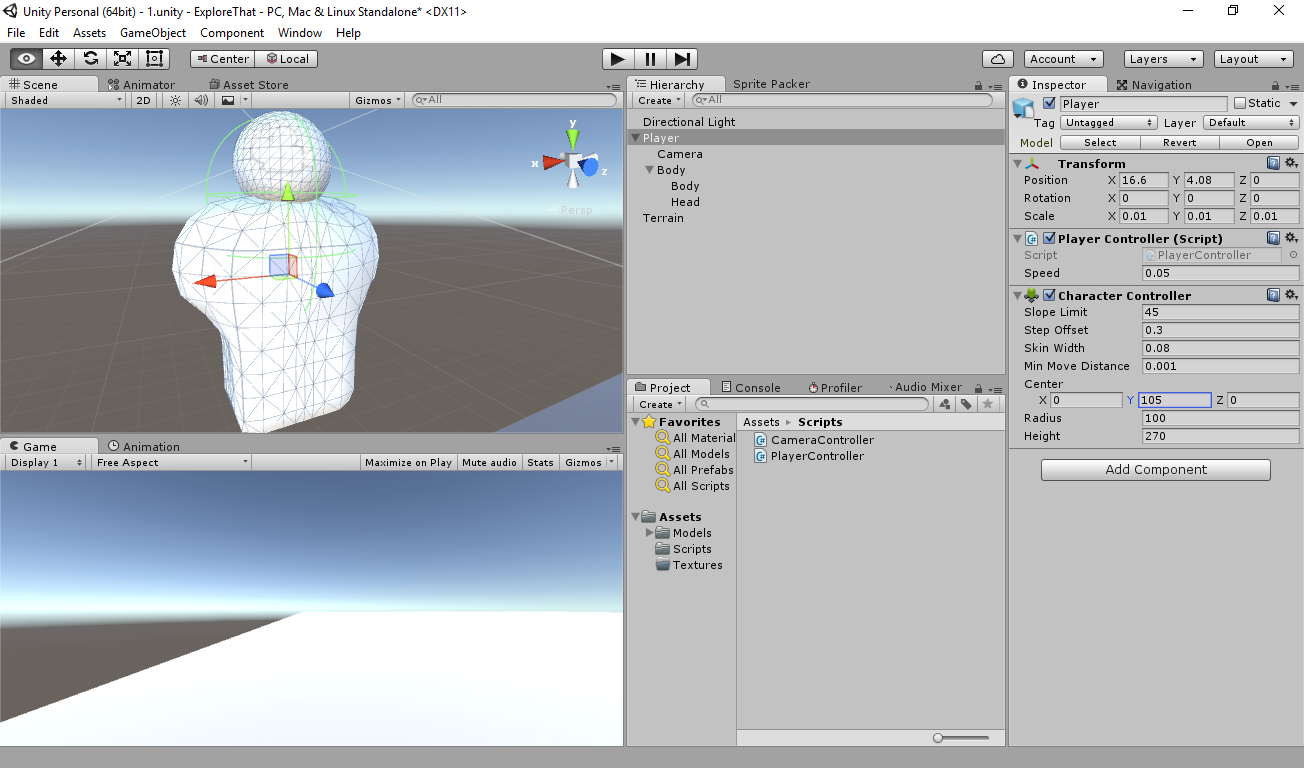


Рисунок – Змінні під час присідання

Частина коду реалізації представлена нижче:

//LEFT CTRL SIT

if (Input.GetKeyDown(KeyCode.LeftControl))

{

character.height = 270;

character.center = new Vector3(0, 105, 0);

speed = 2.5f;

ctrl = true;

}else if (Input.GetKeyUp(KeyCode.LeftControl))

{

character.height = 420;

character.center = new Vector3(0, 34, 0);

speed = 5;

ctrl = false;

}

Варто звернути увагу на те, що до нашого класу я додав приватну логічну змінну ctrl, якою ми будемо показувати, чи на даний момент наший персонаж знаходиться в положенні сидячи. Далі, для реалізації бігу і коректної зміни швидкості і положення нашого персонажу ми введемо булеву змінну shift, яка буде повідомляти нас про те, що персонаж на разі біжить, і заборонити йому під час цього присідати, для цього в умову натискання кнопки додамо значення !shift, яке повідомлятиме про те, що персонаж не біжить. Умова виглядатиме наступним чином:

if (Input.GetKeyDown(KeyCode.LeftControl) && !shift)

Далі нам потрібно реалізувати саму можливість бігу, для цього просто змінюватимемо, а саме - збільшуватимемо швидкість руху, тобто змінну speed, варто не забувати перевіряти, чи персонаж не в положенні сидячи. Приклад коду представлений нижче:

//LEFT SHIFT RUN

if (Input.GetKeyDown(KeyCode.LeftShift) && !ctrl)

speed = 10;

else if (Input.GetKeyUp(KeyCode.LeftShift) && !ctrl)

speed = 5f;

Для того, щоб продовжити роботу варто не забути про гравітацію. До компонету Character Controller гравітація не застосовується, тому при сходженні з гори або при присіданні ми не будемо притягуватися до землі. Для вирішення цієї задачі ми кожен кадр будемо рухати нашого персонажа вниз, для цього до moveDirection додамо вектор, в який у змінну Y ми передаємо глобальне значення гравітації, рядок коду матиме наступний вигляд:

moveDirection += Physics.gravity/7;

Також ми поділили значення на 7, так як всі дії відбуваються дуже різко, а таким чином ми досягли певної плавності дій.

Перейдемо до реалізації стрибка. Оскільки відсутні стандартні методи для реалізації стрибка, тоді ми будемо рухати нашого персонажа вверх, після чого він падатиме під дією «гравітації».

Якщо вирішувати цю проблему простим шляхом, тобто підняти нашого персонажа вверх, то це буде виглядати як різкий ривок і ми не матимемо можливості побачити сам стрибок (рисунок ).

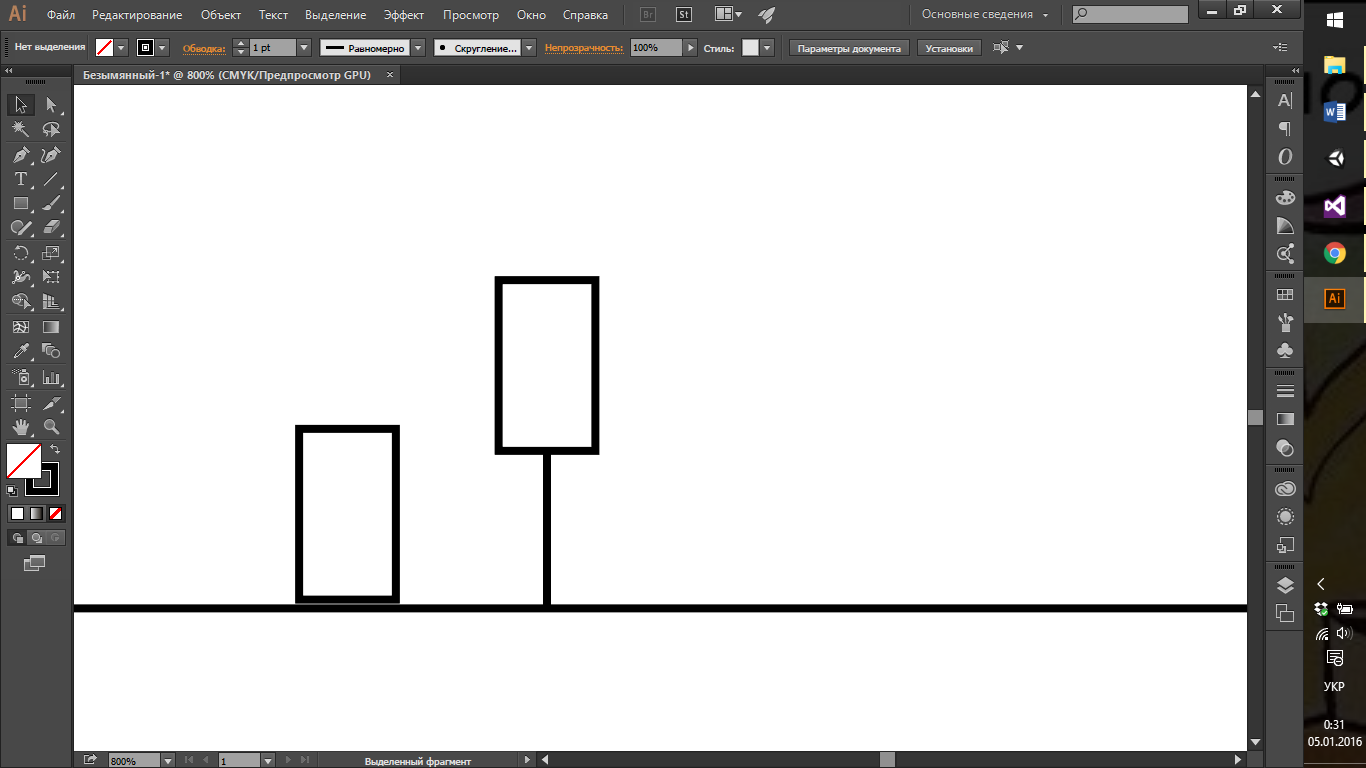
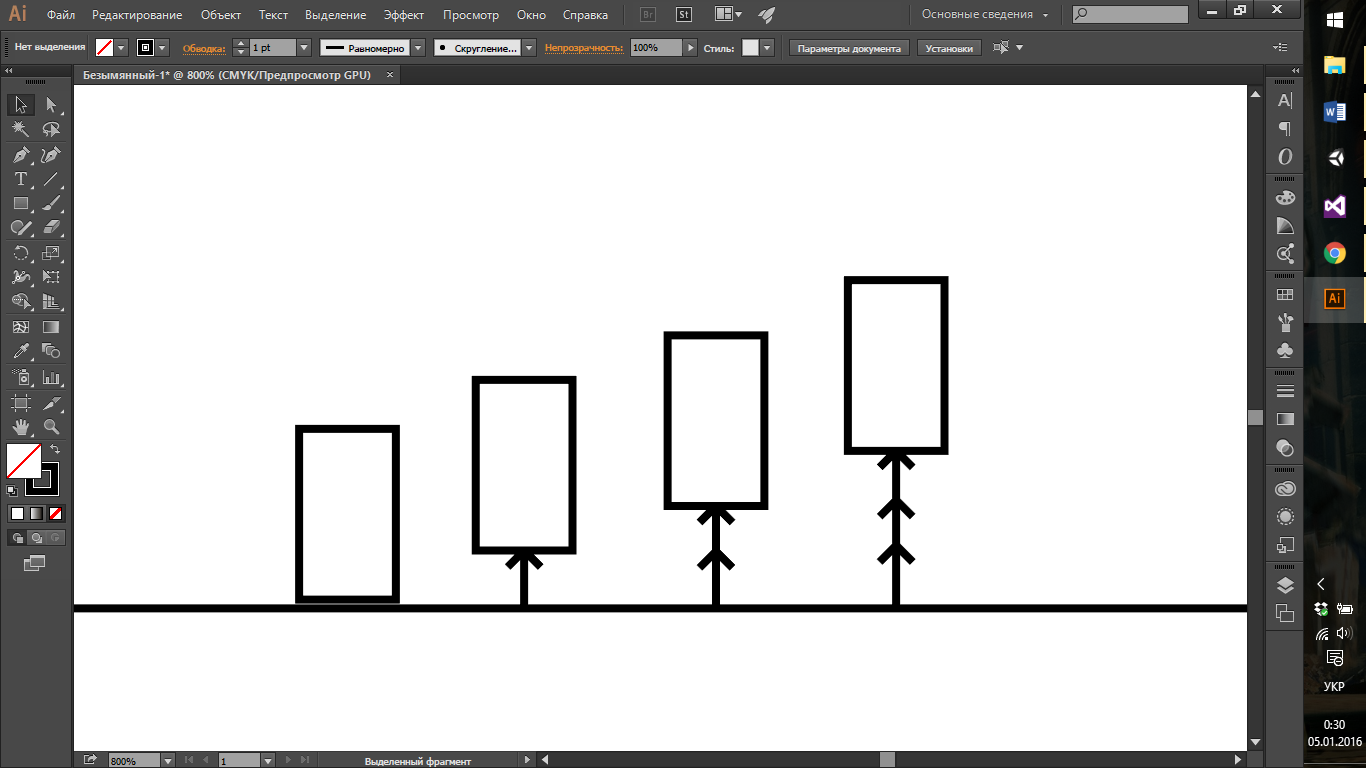


Рисунок – force movement

Тому для вирішення цієї проблеми ми будемо надавати імпульс нашому персонажу кожен кадр, доки він не підніметься на задану висоту (рисунок ).



Рсиунок – поетапне переміщення персонажа

Для вирішення поставленої задачі створимо метод і назвемо його Jump(); і ініціалізуємо змінну типу int, яку назвемо jumpCounter, а також допоміжні змінні jumpDuration і jumpPower, в які будемо записувати довжину стрибка в кадрах, а також силу, яка направлена вверх по вектору.

Тіло методу Jump:

void Jump()

{

if (jumpCounter <= jumpDuration)

{

jumpCounter++;

character.Move(new Vector3(0, jumpPower \* Time.deltaTime, 0));

}

}

А також оголошення змінних для реалізації стрибка:

private int jumpPower = 11,

jumpDuration = 15,

jumpCounter = 21;

Тепер перейдемо в тіло методу Update(), і добавимо строку, в які відслідковуємо натискання на пробіл і добавимо виклик методу для стрибка.

//SPACE JUMP

if (Input.GetKeyDown(KeyCode.Space) && !ctrl && character.isGrounded)

jumpCounter = 0;

Jump();

А також варто передбачити, що під час стрибка гравітація не впливала на персонажа, тому добавимо умову

if (jumpCounter > jumpDuration)

moveDirection += Physics.gravity;

Важливим моментом для вирішення задачі із стрибком є застосування гравітації до перевірки приземлення, тому перенесемо строку з застосуванням гравітації вище.

Виконавши всі дії описані вище, ми отримати клас PlayerController. Код класу представлений нижче:

public class PlayerController : MonoBehaviour

{

CharacterController character;

void Start()

{

character = GetComponent<CharacterController>();//(CharacterController)GetComponent("CharacterController")

}

//movement speed

public float speed = 5;

//forwardDirection that depends on body position

//moveDirection depends from forward direction and view angles

Vector3 forwardDirection,

moveDirection,

sideDirection;

private float yDegrees;

private int jumpPower = 11,

jumpDuration = 15,

jumpCounter = 16;

private bool ctrl = false,

shift = false;

void Jump()

{

if (jumpCounter <= jumpDuration)

{

jumpCounter++;

character.Move(new Vector3(0, jumpPower \* Time.deltaTime, 0));

}

}

void Update()

{

moveDirection = Vector3.zero;

yDegrees = transform.rotation.eulerAngles.y \* Mathf.PI / 180;

forwardDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees));

yDegrees = (transform.rotation.eulerAngles.y + 90) \* Mathf.PI / 180;

sideDirection = new Vector3(Mathf.Sin(yDegrees), 0, Mathf.Cos(yDegrees));

//APPLY GRAVITY

if (jumpCounter > jumpDuration && !character.isGrounded)

character.Move(Physics.gravity \* Time.deltaTime);

//LEFT CTRL SIT

if (Input.GetKeyDown(KeyCode.LeftControl) && !shift)

{

character.height = 270;

character.center = new Vector3(0, 105, 0);

speed = 2.5f;

ctrl = true;

}

else if (Input.GetKeyUp(KeyCode.LeftControl) && !shift)

{

character.height = 420;

character.center = new Vector3(0, 34, 0);

speed = 5;

ctrl = false;

}

//WASD MOVEMENT

if (Input.GetAxis("Vertical") != 0)

moveDirection += Input.GetAxis("Vertical")\* forwardDirection \* speed;

if (Input.GetAxis("Horizontal") != 0)

moveDirection += Input.GetAxis("Horizontal") \* sideDirection \* speed;

//LEFT SHIFT RUN

if (Input.GetKeyDown(KeyCode.LeftShift) && !ctrl)

speed = 10;

else if (Input.GetKeyUp(KeyCode.LeftShift) && !ctrl)

speed = 5f;

//SPACE JUMP

if (Input.GetKeyDown(KeyCode.Space) && !ctrl && character.isGrounded)

jumpCounter = 0;

Jump();

character.Move(moveDirection \* Time.deltaTime);

}

}