**­­Chef Run project**

**Глава 1. King Julian  
«I like to move it, move it»**

1. **Механіки чи онлайн?**

М. Я все ж таки обираю механіки вищим пріоритетом. Обґрунтування: поки я дамі нубик, то важливо зрозуміти що я зроблю всі анімації, предмети і їх connection with players. Також я не думаю що в грі буде багато скріптів, а онлайн режим можна буде створити потім, принаймні так деякі люди так роблять і я зможу.

1. **Запуск**

Unity, Visual S 2022. Остання повна версія юніті і Visual просто йшов з нею.

Далі йшов вибір template. За планом ця гра 3д і по факту можна обирати дефолтне 3д і не маятись з іншим. Але є The URP(Universal Render Pipeline) версія. Вона відрізняється тим що їй +- все одно на платформи і можна буде імпортувати і на пк, і на тел, і на консоль. Мені цікаво як це все робитиметься тому обираю її і сподіваюсь що не буде backfire.

--проєкт запускається більше 3 хвилин, I think I made a mistake \(0-0)/

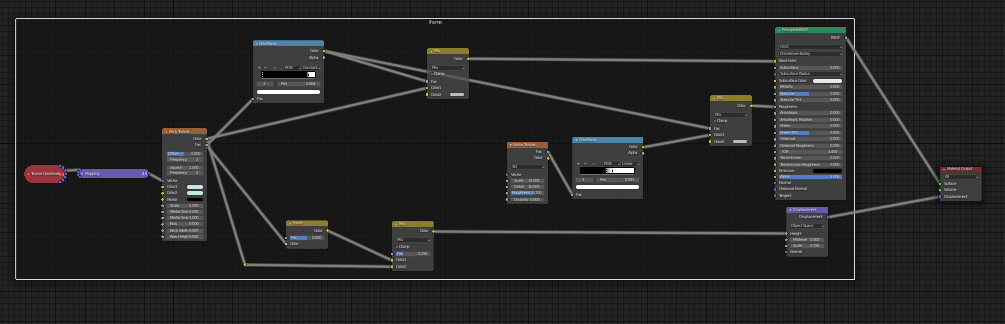
1. **Запуск**

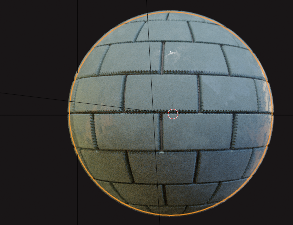
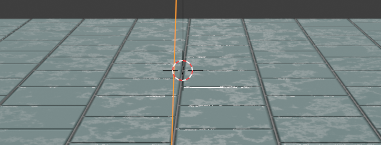
BRUUUHH! Я тіки запустив а вже помилка.



Після 20 хвилин базових налаштувань юніті, преференсів, проєкт менеджерів і URP ассетів.

1. **Неправильний шлях**

Ігри робляться використовуючі все дефолтне без текстур, а гравець – звичайний актор. Але я їбобо і вирішив робити підлогу і миттєво створювати для неї текстуру. Якусь текстуру я не знайшов, тому створення текстури в блендері з нуля.

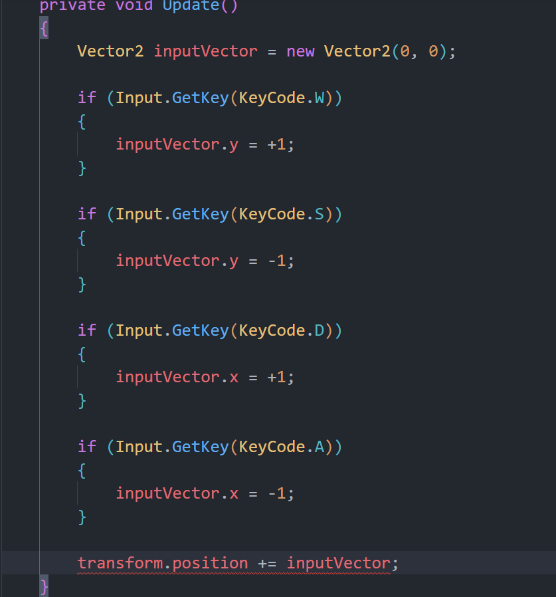
Rougness map була за реалістичною тому тре було її прибрати, щоб красивіши виглядало.



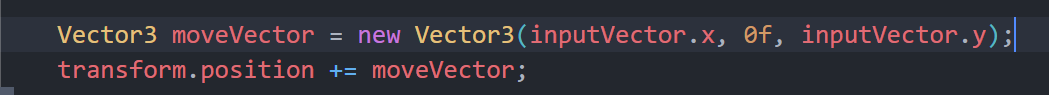
Я наробив ще саме модельок 3д, але зараз в Юніті переносити не буду, бо я задовблюсь з цими всіми матеріалами і так далі, тому перейдемо до писання перших скриптів.

1. **Перші скрипти**

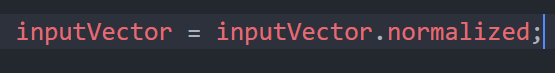
Як рухається гравець? В грі не буде стрибків тому потрібно лише x ta y координати. Якщо людина натискає кнопку гравець рухається в заданому напрямку.

Код виглядає так, але є проблема(в самому низу). Ми не можемо змінити transform.position своїм вектором з двома координатами. Трансформ використовує xyz, a Vector2 – xy.   
  
Fix: створення Vector3 який братиме в себе значення Vector2 і де z=0f

(Ми могли б inputVector зробити вже трьох мірним, але я не думаю що нам потрібно робити саме так, бо ніякого z input в нас не буде)

  
(про збільшену картинку) Тепер вже з цим кодом, гравець ходить в усі напрямки(wasd)

(Я тут відео не додаватиму тому вірити треба мені так.)  
  
Також в нас зустрілись найпопулярніші 2 БАҐИ.   
1-Якщо ми рухаємось по діагоналі, то наш гравець рухається швидше. W(+1;0) D(0;+1). W+D(+1;+1). Ця проблема вирішується доволі легко тим що ми повинні «нормалізувати»

--баґ пофіксили

2-Це те що в юніті все працює на скільки тянить комп, і всі рухи depend on fps. Тобто якщо буде люди гратимуть в гру з 60 фпс в них все буде поовільніше ніж у людей з 120 фпс. Це фікситься за допомогою множника Time.deltaTime(він знаходить скільки часу треба для завантаження 1 фрейму Приклад: 0.001234 секунди. Тим самим це уповільнює все, але робить стабільну швидкість і що у всіх буде однакова швидкість).

До: <https://youtu.be/na0REJvb7cg>

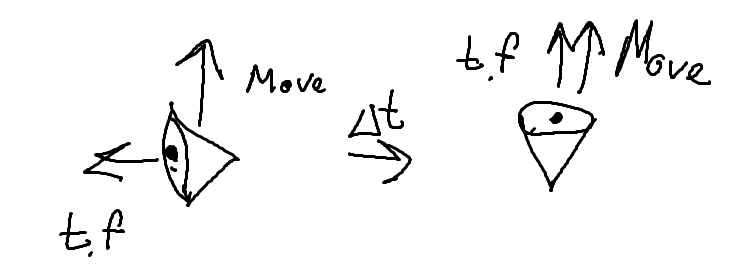
Я тут додаю множник і роблю можливість настройки moveSpeed

Після: <https://youtu.be/eJ4D3KDZrSA>



--це я такого персонажа зробив♥. Імпортую його в форматі .fbx і не забуваю про розмальовані текстури.

Тепер треба зробити так, щоб гравець повертався в ту сторону в яку рухається + він не 1 фрейм поворота але була міні анімація повороту.

Нижче наведена граф. Інструкція як працює Lerp.   
В нас є початкове положення(transform.forward – вектор а) і ми починаємо рухатись «вгору»(moveDirection - b), тоді ми розраховуємо по числу float(Time.deltatime) скільки часу це займає. В данному випадку повертається воно дуууже повільно, тому треба данне число float збільшити.



**Глава 2. No touchy-touchy?**

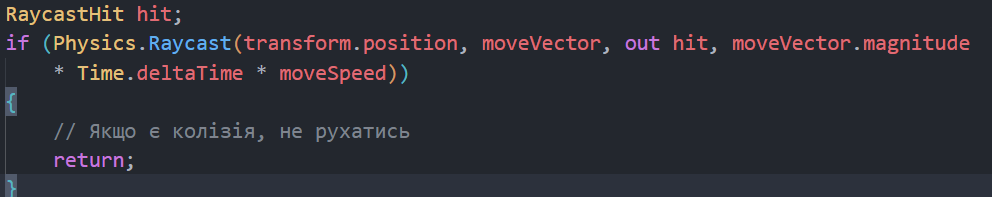
--перед цим я поставив камеру і

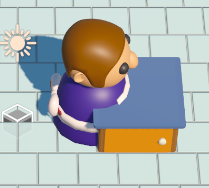
зробив міні adjustments

до кольорів(colors=neutral, contrast++, saturation++),

наче вигляда ок.

Колізія. Знову факап. Я написав цей код:



Але це на від\*\*\*сь зроблено. Коли гравець доторкається до колізії (до цього тре додати Box collider до куба), то він «застрягає». Input людини ігнорується, якщо якийсь напрямок до цієї колізії. Тобто треба відходити від неї, щоб взагалі почати рухатись. Ну і ще воно застрягає наполовину в кубі, бо колізія гравця це точка середини GameObject.

--Висновок – переробити/знайти оптимальний варіант

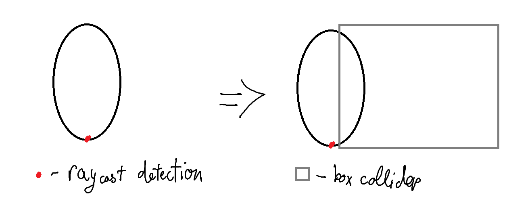
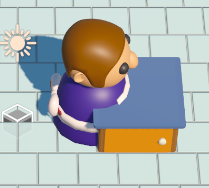
Трішки **теорії** про наш випадок вик-ння. ***Raycast****.* Raycast - це функція для перевірки зіткнень між об'єктами в 3D просторі. Вона дозволяє визначити, чи перетинається промінь з певною початковою точкою і напрямком з іншими об'єктами, які можуть мати коллайдери.

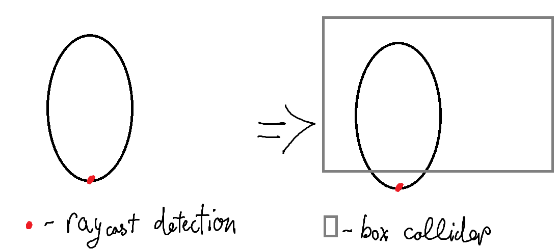
Функція приймає три аргументи:

* **origin** - початкова точка, з якої починається промінь. Це може бути точка у просторі, де знаходиться об'єкт або його частина. (в нашому випадку, коли гравець рухається transform.position)
* **direction** - напрямок променя. Це може бути вектор, що показує, куди спрямований промінь. працюватиме (куди рухатимться гравець moveVector)
* **distance** - дальність, яку буде проходити промінь в заданому напрямку. (тут вже ми в коді повинні описати на скільки широке пузіко у персонажа і на скільки в нього thicс бедра)

Якщо **промінь зіткнувся** з яким-небудь об'єктом, функція поверне **true**, в іншому випадку - **false**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Через painful amount of time reading reddit та юніті форми, я дізнався, що одна проблема якраз у Raycast. Ми вже бачили на цій картинці, що плеєр ніби «їсть» цей ящик. Саме **RAY**cast бачить внизу об’єкта ніби точка і рисунок нижче для нього це норма, бо саме червона крапка не торкається колізії і все одно на модель. Так ми можемо записати дистанцію, але все одно це буде коряво. БО В **RAY**cast нема ще позначки висоти і ми можемо побачити візуалізацію в наступному малюнці.



--Це теж буде для нього ок і плеєр зможе ходити крізь куб.

Тобто в данному випадку ми можемо використовувати CapsuleCast(капсула а не бокс чи сфера, бо акторами/player є капсули і наш перс +- схожий на циліндр)

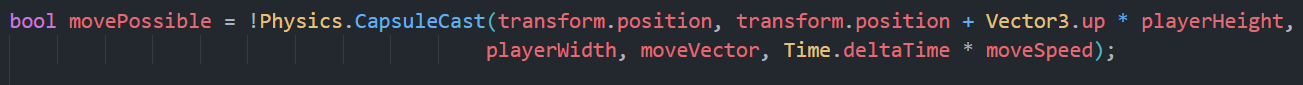
Згадаємо теорію, що **CapsuleCast** повертає **BOOLEAN.** Тобтонам треба зробить, що коли не зустріли колізію то рух можливий, якщо зустріли то не рухаємся.(логіка дуже схожа з кодом який я написав спочатку, але тепер можна зробити по іншому типу:)

if(movePossible)

{//можна рухатись змінюючи transform.}

if (!movePossible || // але гравець натискає клавіші, які рухають його по діагоналі)

{//то слайдини по кубу}



У даному випадку, параметр **transform.position** визначає **позицію початку** капсули, а **transform.position + Vector3.up \* playerHeight - кінцеву позицію**.

--Vector3.up - вектор, що показує напрям вгору від глобальної координатної системи.

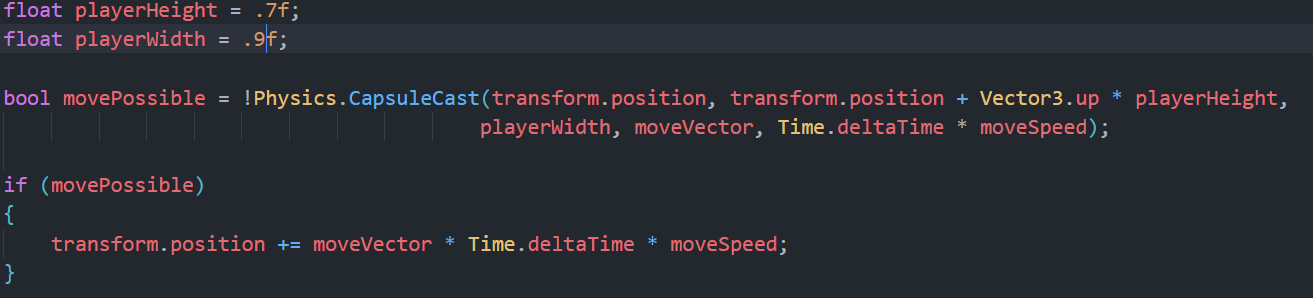
**playerHeight** - це **висота гравця**.

**playerWidth** - це **ширина капсули**.

**moveVector** - вектор, який вказує **напрямок руху** гравця.

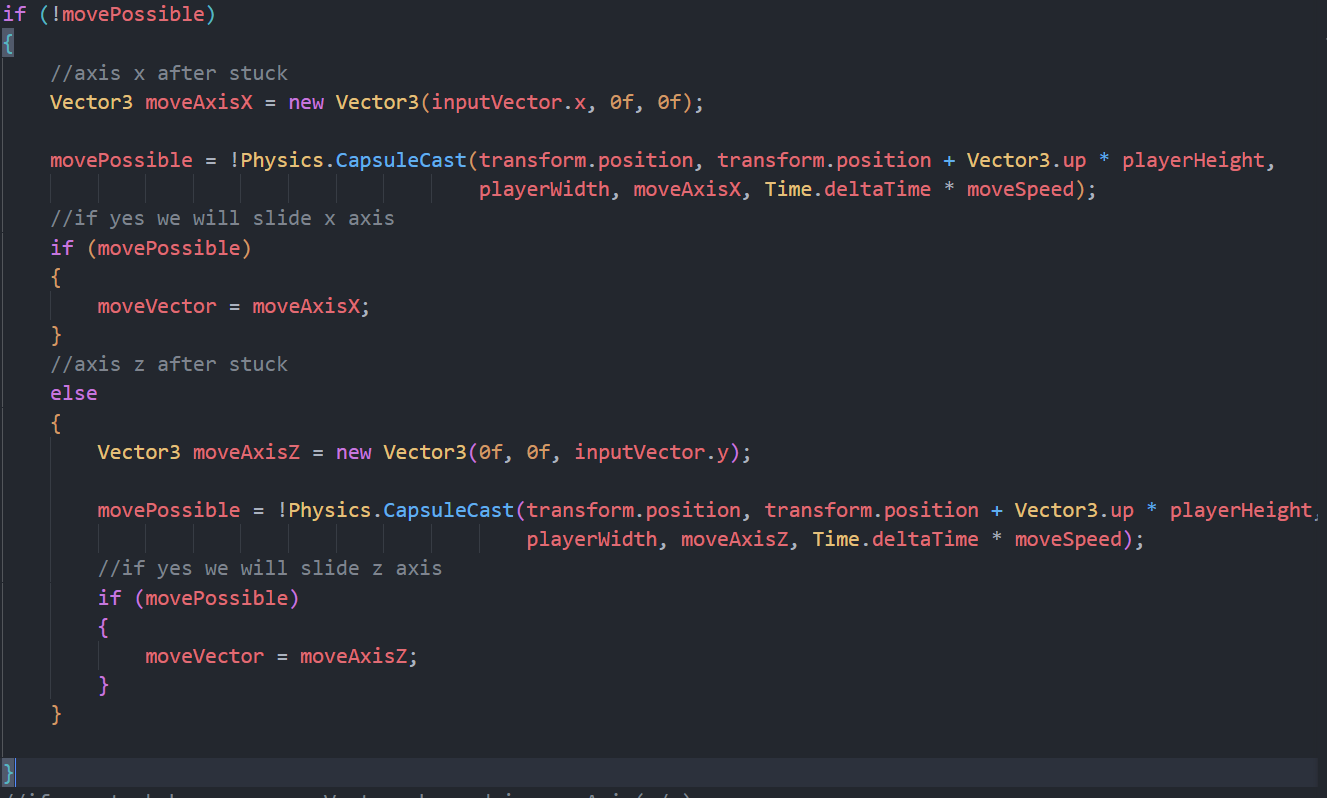
Якщо метод CapsuleCast повертає значення true – немає колізій з іншими об'єктами, тобто рух можливий (movePossible = true). Якщо false, то наявність колізій (movePossible = false).

Результат:

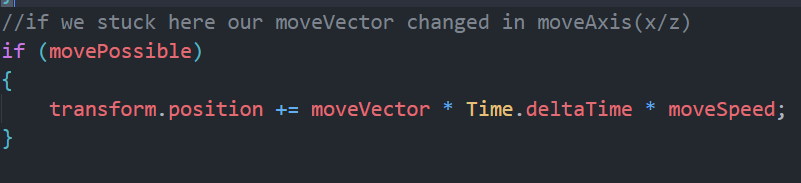
-- <https://www.youtube.com/watch?v=-EV0qRQEXG8> PlayerWidth підбирав підбором, я думав воно буде як і координати, але ні.

Зараз, коли торкаюсь колізії я не можу слайдити, а просто ніби застрягаю там. Ідентична ситуація першої помилки. Але тепер настав час додати фічу слайдінга.

Як я розумію то ситуація: Ми не можемо рухатись бо вперлись => тоді перевіряємо ч можемо рухатись ліворуч-праворуч, колізії не знайдено, нам можна рухатись ліво-право.  
або  
=> перевіряємо чи можемо рухатись вгору-вниз, колізії не знайдено, нам можна рухатись.

І типу це все запхати, в те що коли movePossible = false.

--Це код для рухання з колізіями.

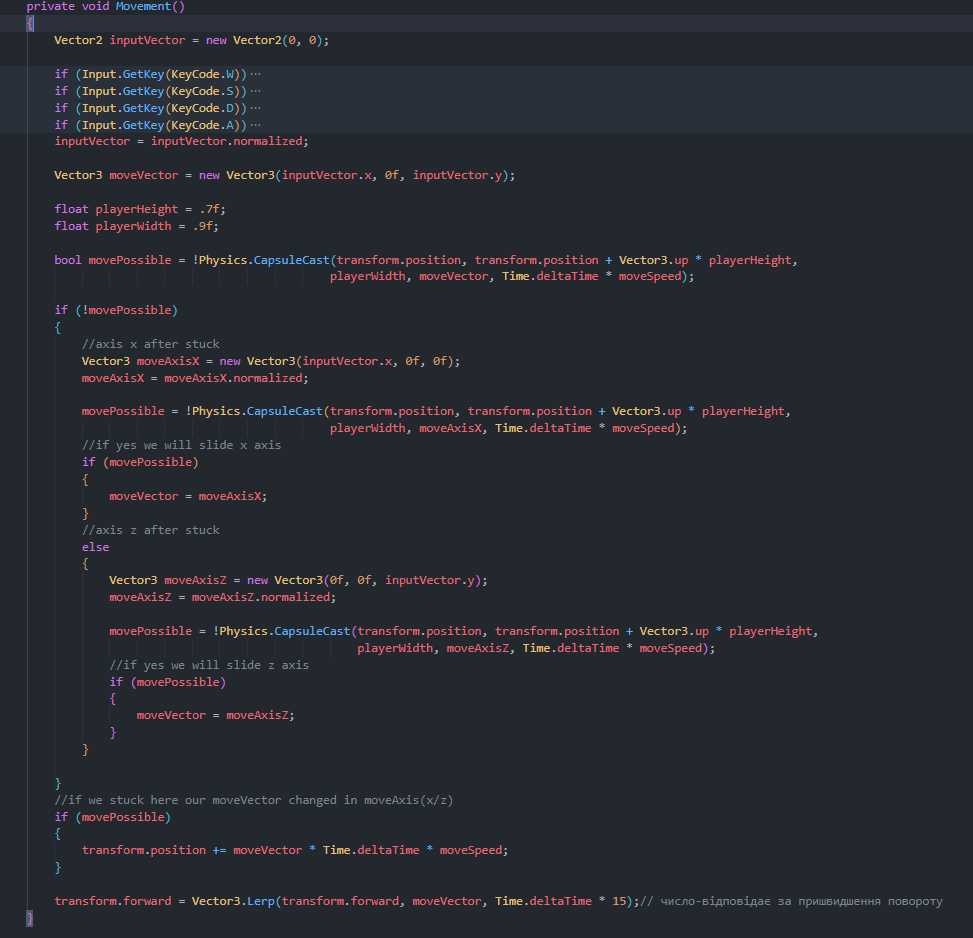
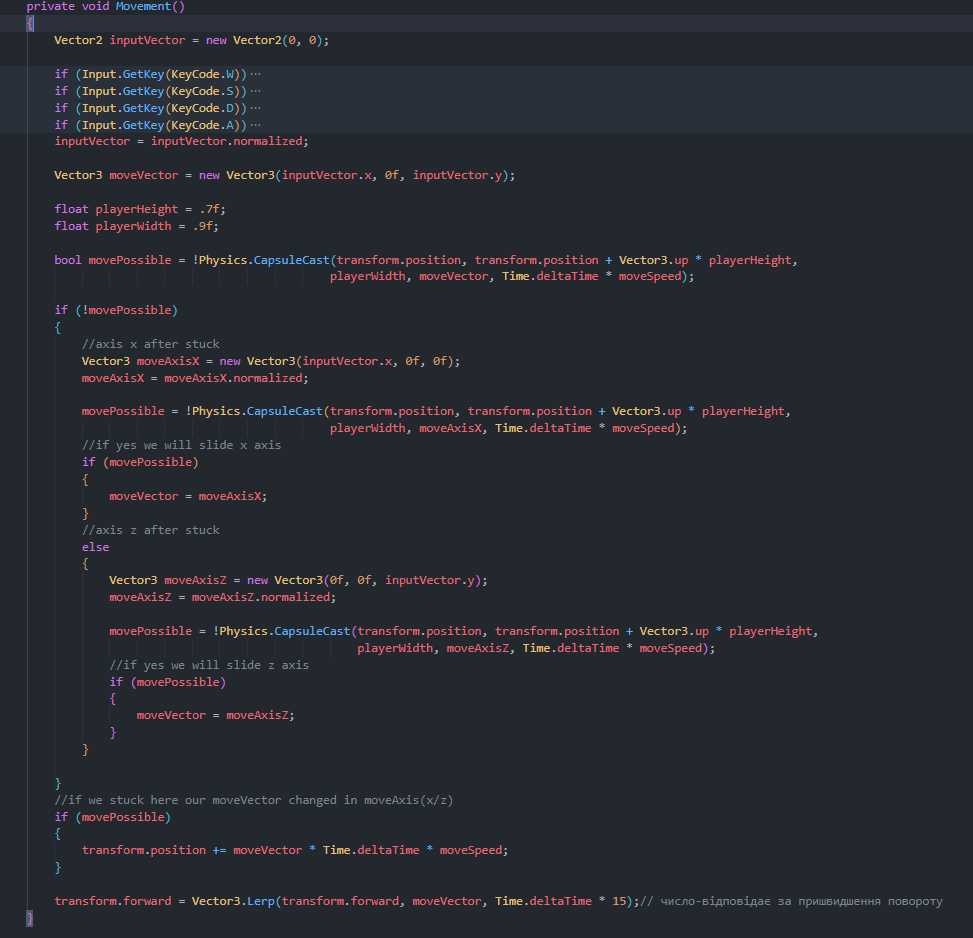
Пояснення. Якщо ми зіткнулись з колізією(movePossible=fallse). Створюємо новий вектор3, який відповідатиме за рух по якійсь axis(x чи z). Далі ми переписуємо movePossible і замінюємо там direction(moveVector->moveAxisX). Тепер ми знову перевіряємо чи можемо рухатись, якщо так то ми слайдимо по цій axis. Х та Z абсолютно ідентичні крім назв векторів і положення inputVector.x та у.

Результат: https://www.youtube.com/watch?v=TS8axpqKRHc

\*цього нема в цих картинках і результаті\*

Але, можна помітити що як і в перших кроках, по діагоналі він рухається з іншою швидкістю, фікситься це так само треба прописати

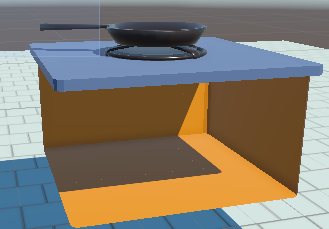
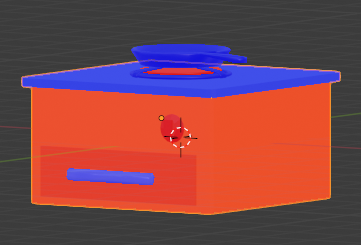
«…moveAxisX = moveAxisX.normalized; ….

….moveAxisZ = moveAxisZ.normalized;….»

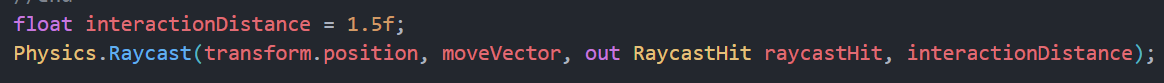
**Глава 3. If you touch you will turn into a Ligma!**

**Короткий опис частини**: Що перед гравцем? Логіка: гравець рухається в cupboard і тоді він з нею може interact, якщо він зупинився і не рухається більше, то ми заносимо в змінну останній об’єкт з яким можливе було interact. Далі розписано реалізацію в юніті, бо це не тільки 1 код.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bug fix:

--Юніті відображає лише фронт частину Faces. Тому перед переноскою у Юніті з Блендера, треба перевірити як направлені всі фейси, тобто щоб нормально відображалось тре все синє.

Знову повертаємось до складний Raycast.

 Використовуючи метод Physics.Raycast, перевіряється наявність перешкод в напрямку, в якому гравець намагається взаємодіяти. Аргументи :

- позиція початку перевірки (transform.position),

- вектор напрямку перевірки (moveVector),

- довжина луча перевірки (interactionDistance – 1.5)

- параметр raycastHit, який зберігає результати перевірки.

*--just saying:*

*дані що зберігає цей raycastHit:*

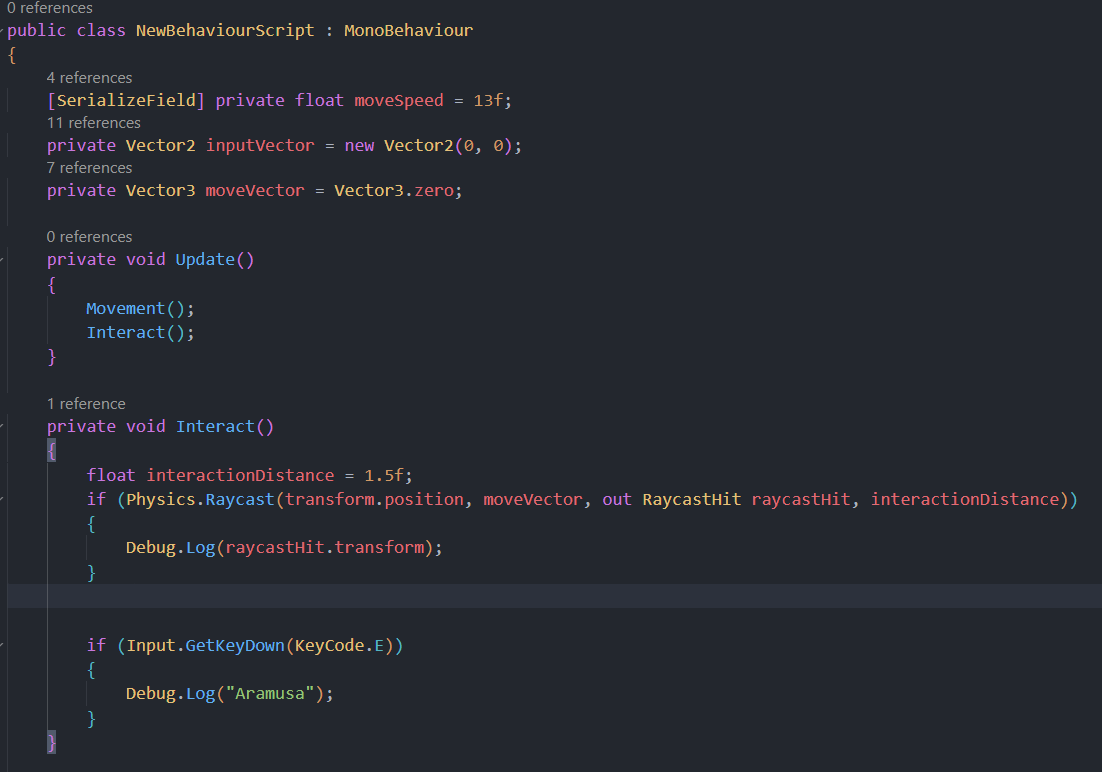
*точка зіткнення, вектор нормалі в точці, дистанція і*

*який самий колайдер поціловано. Ми визначаємо тип перешкоди*

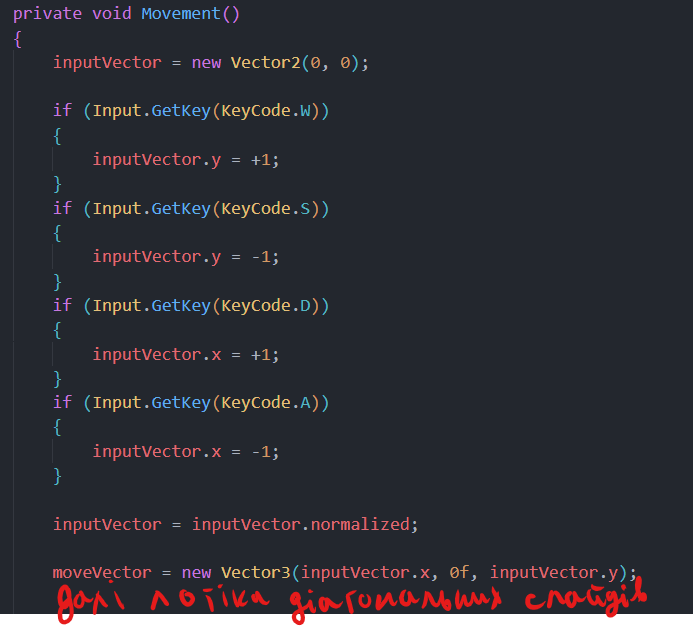
*за допомогою цього, буде це різні cupboard-и чи ще щось. Може*

*потім я дойду до зіткнень між людьми, але поки це на цьому місці.*

Початок практики:

Оскільки нам в фу-ї Interac() потрібні значення moveVector i inputVector з ф-ї Movement(), ми їх зробили глобальними в цьому класі --хз на скільки це грамотно-правильно-- , але в такому випадку нам треба копіпастити велику частину коду. moveVector та inputVector значення додаютться в Movement, а потім ми можемо використовувати їх в Інтеракт().

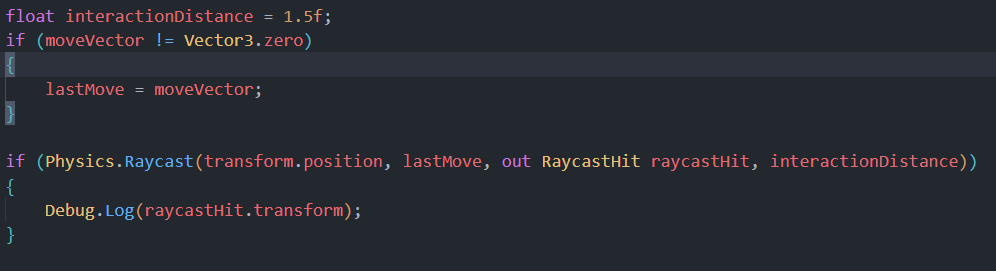
Але якщо, ми захочемо зробити в функціях ці вектори якось УНІКАЛЬНИМИ, то полізуть баґи. Але фіксити це все буду вже майбутній я.

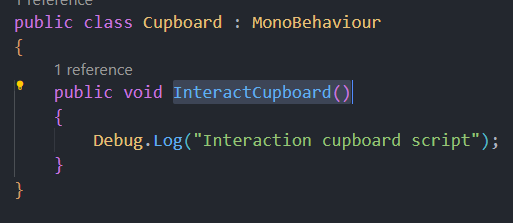


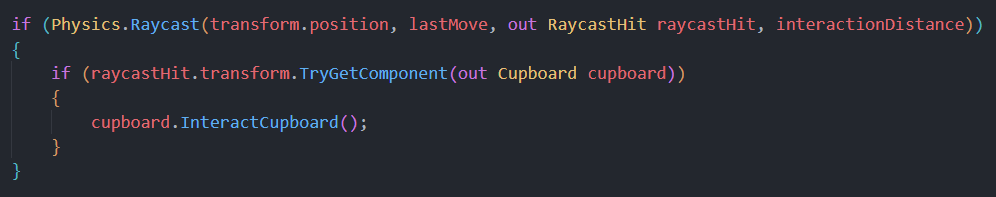
--Тут просто кі-тьь коду зменшелась бо значення переносяться нагору. Єдине що мені здалося, що player став рухатись повільніше, але чогось я впевнений, що це я собі надумав таке.

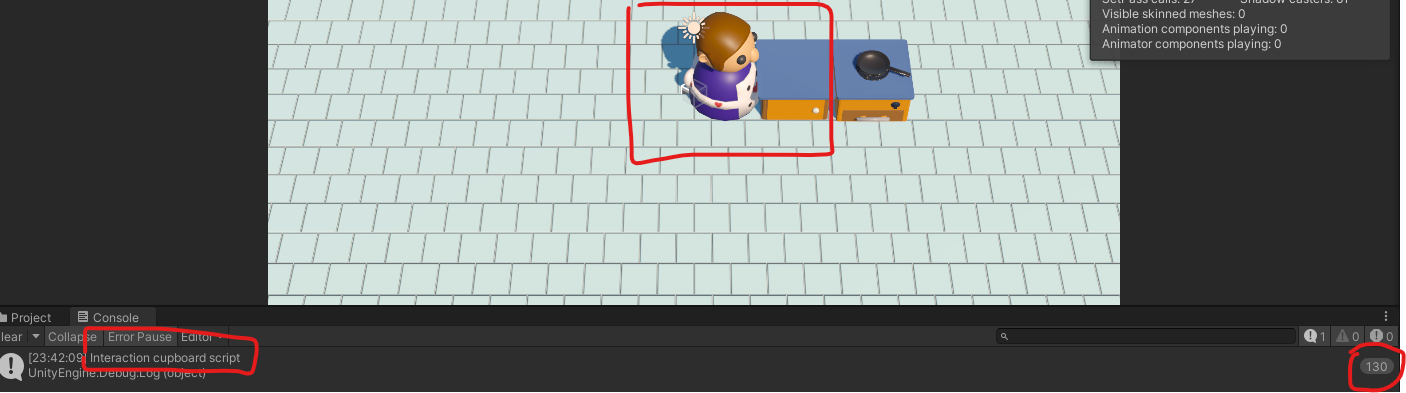
--Фан факт в знайденому туторіалу по інтерактам вказали на те що при використанні вектору(в нашому випадку moveVector), то тоді коли гравець не рухатиметься, то і за такою логікою перед ним нічого не стоятиме, бо вектор = (0 0 0).

Гот фікс:

в нас є глобальна змінна(lastMove), яка зберігає останній напрямок руху. І тоді плеєр зможе «бачити» об’єкт перед ним, коли гравець не натискатиме на клавіші.

Тепер треба створення скрипта для самого cupboard. Простий скрипт який визиватиметься плеєром. Тому паблік.

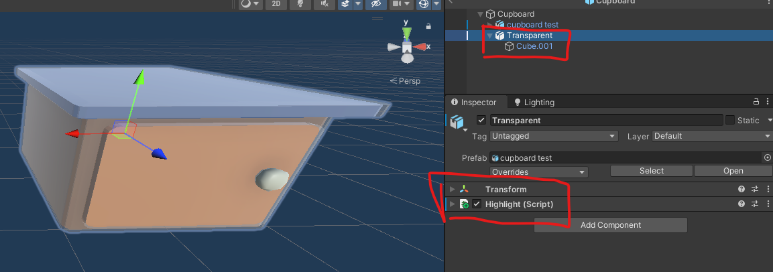
Нижче наведений код з скрипта Плеєра. Там ми перевіряємо чи взаємодіємо з компонентом Cupboard і якщо так то визиваєм його ф-ю



**Глава 4. Meridia’s blessing**

Наш плеєр вже бачить і розрізняє різні типи Cupboard, тепер треба додати візуальну частину(highlight), щоб і сам гравець бачив куди дивиться плеєр.

Створюємо папку з Префабами і там додаємо «нову» модель до звичайного Cupboard. Ця модель це трішки збільшений cupboard але з матеріалом білої альфи.

Ось так воно виглядає, ніби обраним. А далі ми прикріпимо новий скріпт і там розпишимо функціонал.

|  |  |
| --- | --- |
| Інфо | Непотрібний коментар. |
| Оскільки сам **код** і його **переробка змінювались** на цьому етапі і ще це займало декілька днів – я вирішив, що в цій главі я знову розпишу увесь код і його значення без описування «процеса» як раніше. Але це буде вже в кінці глави. | -- наприклад я зрозумів що так званий Cupboardтреба якось перейменувати бо ще буде мінімум 2 таких схожих об’єкта. Фан факт: я не зміг сам доперти до назви тому я описав чату, що це cupboard без нічого і він більш нічим не виділяється і їх відповідь була ну назви його Clear. !-\_-! Чому я сам до цього не додумався?! |

BugS

--оскільки в мене є щось схоже на дислексію, я не міг знайти помилку бо для мене це однаково написаних слова було. AAAAAH!!!

Я начебто створив функціонал higlight, але є 2 великих і значних баґи пов’язані з ним.

1 – Воно мігає і дратує. Воно повинно просто загоратись і залишатись таким, але не мінятись туди-сюди

2 – Іноді коли відходиш в протилежному напрямку чи різко право-ліворуч, то highlight залишається а Interaction неможливо, що показано кліканням на E.

https://youtu.be/K\_bAYnvUJ6E

Рішень може бути багато і які я не знаю. На 1 можна спробувати А – збільшити меш гайлайту. Бо матеріали можуть кілпатись. Б – додати LayerMask до Raycast.

На 2 баґ, я фікса не знаю тільки передивитись/переробити увесь код(.

**Вирішилось:** наново переписано тими же символами і буквами, може щось було з не тією мовою чи ще щось такого типу. І додано купу однакових перевірок і обнульовувань гайлайту об’єкта.

Результат: <https://youtu.be/IIciv9hFvKU>

А тепер настав час розібрати увесь наявний код.

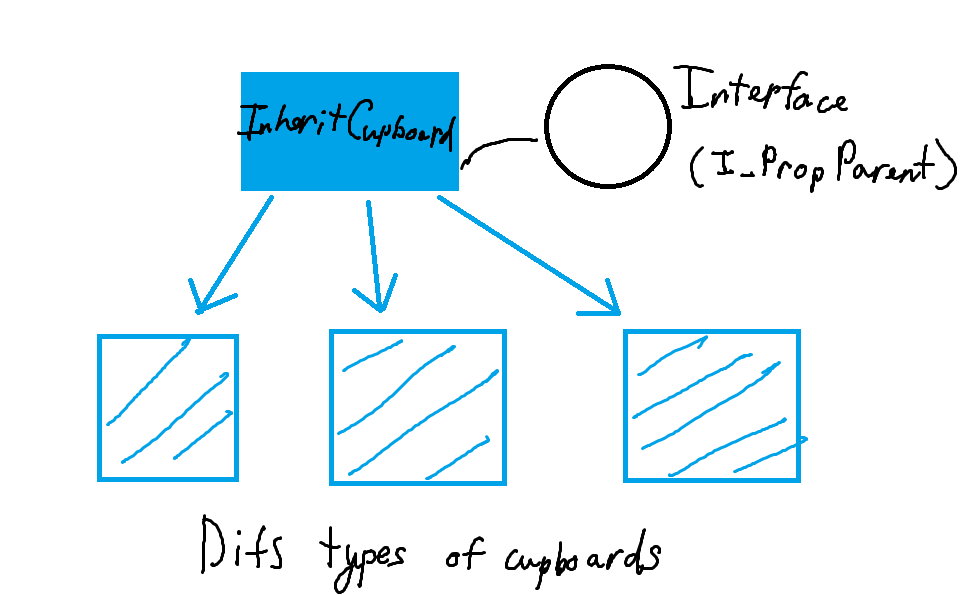
|  |  |
| --- | --- |
| **Код** | **Мій коментар/пояснення** |
| using System;  using System.Collections;  using System.Collections.Generic;  using UnityEngine; | Почнемо спочатку, я вноситиму в табл з кольорами тупо скоопійовано з Visual Studio, бо так буде красивіше і особисто мені зрозуміліше.  Тут додано System. System включає базові класи C#, такі як EventHandler та EventArgs. Які ми використовуємо для гайлайтів на даний момент. |
| public class Player : MonoBehaviour  {      //hughlight stuff      public static Player Instance { get; private set; } | public static Player Instance { get; private set; }  Це статична властивість, яка дає доступ до **єдиного** екземпляру класу Player. *– тобто оскільки я роблю ОНЛАЙН гру то це костиль і його треба буде змінювати якось. Але поки що мені для однієї людини б зробити.*  Тут є один розумний момент що **СЕТТЕР ПРИВАТНИЙ,** тобто ми встановлюємо його лише тут. А інфу зможемо брати з іншого скріпта який відповідає якраз за підсвічування. |
| private void Awake()      {          if (Instance != null)          {              Debug.Log("Instance != null. Error");          }          Instance = this;      } | Якщо інший екземпляр вже існує, це означає, що виникла помилка, і – запис консолі.  Якщо ж інший екземпляр не існує, цей екземпляр буде встановлений як єдиний екземпляр із допомогою статичного поля Instance. Це означає, що будь-який інший об'єкт в програмі може отримати доступ до цього єдиного екземпляру, використовуючи це поле.  --Важливо розуміти що без присвоєння екземпляру гра крашиться і видає помилки що object reference not set to an instence object |
| public EventHandler<OnHighlightCupboardEventArgs> OnHighlightCupboard; | Використовуємо подію. Подія - це механізм, який дозволяє класу повідомляти про інформаційні зміни, що відбуваються в ньому, і забезпечує зв'язок між об'єктами. Ми його використовуємо ще в скрипті до гайлайту, але до цього ми дойдемо після скрипта Player |
| public class OnHighlightCupboardEventArgs : EventArgs      {          public ClearCupboard selectedCupboard;      }      //end highlight stuff part 1 | Ця частина відповідає за обирання cupboard для підсвічування і необхідні дій з ними.  Note\*ми тут саме використовуємо класс, бо без нього ми не змогли б отримати інфу в інший скрипт. Клас OnHighlightCupboardEventArgs визначає тип аргументу, який буде переданий разом з подією. В даному випадку, цей клас містить властивість selectedCupboard, яка містить інформацію про виділений cupboard. Цей клас дозволяє передавати цю інформацію разом з подією. |
| //layerMask that player could interact with only markeed objects as cupboards \*note it is also must be done in unity ui  [SerializeField] private LayerMask cupboardsLayerMask;      [SerializeField] private float moveSpeed = 13f;      private Vector2 inputVector = new Vector2(0, 0);      private Vector3 moveVector = Vector3.zero;      private Vector3 lastMove = Vector3.zero;      private ClearCupboard selectedCupboard; | Тут це дані, які я думаю, зрозумілі і просто з назв |
| private void Update()      {          Movement();          Interact();      } | Ubdate по юнітські визивається кожен фрейм. |
| private void Movement()      {          inputVector = new Vector2(0, 0);          if (Input.GetKey(KeyCode.W))          {              inputVector.y = +1;          }          if (Input.GetKey(KeyCode.S))          {              inputVector.y = -1;          }          if (Input.GetKey(KeyCode.D))          {              inputVector.x = +1;          }          if (Input.GetKey(KeyCode.A))          {              inputVector.x = -1;          }          inputVector = inputVector.normalized;          moveVector = new Vector3(inputVector.x, 0f, inputVector.y);          float playerHeight = .7f;          float playerWidth = .9f;          bool movePossible = !Physics.CapsuleCast(transform.position, transform.position + Vector3.up \* playerHeight,                                                  playerWidth, moveVector, Time.deltaTime \* moveSpeed);          if (!movePossible)          {              //axis x after stuck              Vector3 moveAxisX = new Vector3(inputVector.x, 0f, 0f);              moveAxisX = moveAxisX.normalized;              movePossible = !Physics.CapsuleCast(transform.position, transform.position + Vector3.up \* playerHeight,                                                  playerWidth, moveAxisX, Time.deltaTime \* moveSpeed);              //if yes we will slide x axis              if (movePossible)              {                  moveVector = moveAxisX;              }              //axis z after stuck              else              {                  Vector3 moveAxisZ = new Vector3(0f, 0f, inputVector.y);                  moveAxisZ = moveAxisZ.normalized;                  movePossible = !Physics.CapsuleCast(transform.position, transform.position + Vector3.up \* playerHeight,                                                      playerWidth, moveAxisZ, Time.deltaTime \* moveSpeed);                  //if yes we will slide z axis                  if (movePossible)                  {                      moveVector = moveAxisZ;                  }              }          }          //if we stuck here our moveVector changed in moveAxis(x/z)          if (movePossible)          {              transform.position += moveVector \* Time.deltaTime \* moveSpeed;          }          transform.forward = Vector3.Lerp(transform.forward, moveVector, Time.deltaTime \* 15);// число-відповідає за пришвидшення повороту      } | Moovement ми розбирали раніше, тому я не бачу сенсу ще раз про це повторювати. Але код просто тут залишу |
| private void Interact()      {          float interactionDistance = 2f;  //lastMove needed if our player stand still and we want that interaction would be possible          if (moveVector != Vector3.zero)          {              lastMove = moveVector;          }          if (Physics.Raycast(transform.position, lastMove, out RaycastHit raycastHit, interactionDistance, cupboardsLayerMask))          {              if (raycastHit.transform.TryGetComponent(out ClearCupboard clearCupboard))              {                  //interaction after key input                  if (Input.GetKeyDown(KeyCode.E))                  {                      clearCupboard.InteractCupboard();                  }                  //highlight                  if (clearCupboard != selectedCupboard)                  {                      SetSelectedCupboard(clearCupboard);                  }              }              else              {                  // Check if the player is leaving the selected cupboard area                  if (selectedCupboard != null)                  {                      SetSelectedCupboard(null);                  }              }          }          else          {              // Check if the player is leaving the selected cupboard area              if (selectedCupboard != null)              {                  SetSelectedCupboard(null);              }          }      } | Interact. Повторю, що тут ми дізнаємося що перед плеєром, який об’єкт і якщо це звичайний clearCupboard то ми його помічаємо selectedCupboard, щоб потім додати на нього highlight.  if (Physics.Raycast …..  ми використовуємо метод Raycast з модуля Physics, щоб визначити, чи є об'єкт, на який спрямований останній рух гравця. Якщо промінь зіткнувся з об'єктом зі шару cupboardsLayerMask, то ми продовжуємо перевірку.  Далі, в рядку другому, ми використовуємо метод TryGetComponent, щоб перевірити, чи має об'єкт компонент ClearCupboard.  У рядку третьому ми перевіряємо, чи є об'єкт clearCupboard, тим сами чи ні. Якщо об'єкт clearCupboard не є обраним об'єктом, то ми викликаємо метод SetSelectedCupboard(clearCupboard).  Далі ми в інших випадках обнуляєм selectedCupboard.  Далі буде метод SetSelectedCupboard(clearCupboard). |
| private void SetSelectedCupboard(ClearCupboard selectedCupboard)      {          this.selectedCupboard = selectedCupboard;          OnHighlightCupboard?.Invoke(this, new OnHighlightCupboardEventArgs          {              selectedCupboard = selectedCupboard          });      } | Перший рядок методу встановлює вибраний cupboard на об'єкт, який передається як аргумент методу. У цьому випадку, метод отримує об'єкт "selectedCupboard".  Другий рядок методу викликає подію "OnHighlightCupboard" (яка, як ми розглядали раніше, є типом "EventHandler<OnHighlightCupboardEventArgs>"), якщо вона не є рівною null (позначається знаком питання ?). Ця подія використовується для відображення вибраного cupboard в грі. |

**Тепер розглянемо скрипт Highlight**

|  |  |
| --- | --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class HighlightCupboard : MonoBehaviour  {      //for insert prefabs and visual      [SerializeField] private ClearCupboard clearCupboard; //prefab of cupboard visual      [SerializeField] private GameObject highlightObject; //prefab of transparent visual      private void Start()      {          Player.Instance.OnHighlightCupboard += Instance\_OnHighlightCupboard;      }      private void Instance\_OnHighlightCupboard(object sender, Player.OnHighlightCupboardEventArgs p)      {          if (p.selectedCupboard == clearCupboard)          {              highlightObject.SetActive(true);          }          else          {              highlightObject.SetActive(false);          }      }  } | --фан факт: нам треба щоб Instance з скрипта Player активувалось перед частиною Player.Instance…. з цього скрипта. Тому там ми використовуємо awake а тут start.    Коли в Player станеться якась подія, він може повідомити про це. Player використовує обробник подій OnHighlightCupboard для інформування всіх підписаних на цю подію.  У випадку HighlightCupboard, метод Start() встановлює обробник подій Instance\_OnHighlightCupboard для події OnHighlightCupboard в класі Player. Це дозволяє HighlightCupboard реагувати на зміни в Player.  Метод Instance\_OnHighlightCupboard() є обробником події OnHighlightCupboard. Цей метод приймає об'єкт-відправник і екземпляр OnHighlightCupboardEventArgs, який містить вибраний cupboard гравцем. В методі перевіряється, чи обраний cupboard є поточним clearCupboard, і якщо так, то активується об'єкт highlightObject, що відповідає за виділення cupboard. Інакше highlightObject деактивується. |

**Глава 5. «Pick it Up. PICK. IT. UP»**

***--****описую я коротко тут без поглиблення і тільки код, без юніті і всіх прив’язок [SerializeField].*

****В цій Главі ми використовуємо **інтерфейси, Inheritance(**наслідування**),** і зміну **parents.**

В нас є скрипт(InheritCupboard) до якого підключений інтерфейс(I\_PropParent). І є різні типи cupboards, які наслідують за першим скриптом, але в різних типах прописані різна логіка. Наприклад контейнер спавнить об’єкт, а з пустого можна брати об’єкт чи ставити.

Як це виглядало в кінці:  
https://youtu.be/UzB2X9urgYA

Коротко розберемо вже знову перероблений код.

Почнімо з інтерфейсу:

|  |  |
| --- | --- |
| public interface I\_PropParent  {      //here lays setters, getters, the one which make null and boolean to understand is there prop at all.      public Transform GetPropTransform();      public void SetPropParent(Props prop);      public Props GetProp();      public bool HasProp();      public void NullProp();} | Хто імплімінтує цей інтерфейс матиме такі 5 методів. В нашому випадку матимуть цей інтерфейс InheritCupboard і Player (Бо предмети ми беремо з cupboards у руки player, тобто змінюємо parents. \*це не робиться в інтерффейсі, а робиться ЗА ДОПОМОГОЮ його, бо ми розділяєм дані між тими хто його імплементив) |

Тепер InheritCupboard з якого наслідую всі інші cupboards

|  |  |
| --- | --- |
| public class InheritCupboard : MonoBehaviour, I\_PropParent  {      [SerializeField] protected Transform topPoint; //point where things will spawn/carried etc      //Props class which contain info of PropSort where we connected cheese/onion/tomato etc      //we need this so player understand what object is not just simple PROP but a TOMATO.      protected Props prop;      public virtual void InteractCupboard(Player player)      {      }      public Transform GetPropTransform()      {          return topPoint;      }      public void SetPropParent(Props prop)      {          this.prop = prop;      }      public Props GetProp()      {          return prop;      }      public bool HasProp()      {          return prop != null;      }      public void NullProp()      {          prop = null;      }} | Клас містить метод InteractCupboard, який приймає об'єкт гравця та буде override в класах-спадкоємцях, щоб забезпечити різні функціональність при взаємодії з cupboard. |

Я думаю, що по коментарям написаних в коді +- достатньо для розуміння приблизно що відбувається.

Далі роздивимся два схожих кода і наслідників цього класу. Container = spawner, ClearCupboard = put/take object.

|  |  |
| --- | --- |
| public class Container : InheritCupboard  {      [SerializeField] private PropsSort propSO; //so = scriptable object      public override void InteractCupboard(Player player)      {          //check that player can pick up only one object          if (!player.HasProp())          {              //creation of prop prefab in topPoint              //and link the prop to a clearCupboard object              Transform propTransform = Instantiate(propSO.prefab, topPoint);              propTransform.GetComponent<Props>().SetPropParent(player);          }      }  } | public class ClearCupboard : InheritCupboard  {      [SerializeField] private PropsSort propSO; //so = scriptable object      public override void InteractCupboard(Player player)      {          //check is there any object on a cupboard          if (!HasProp())          {              //no prop found so check if player holding smth drop here              //else do nothing              if (player.HasProp())              {              player.GetProp().SetPropParent(this);              }          }          //so there is smth in cupboard. If player has smth on hands do nothing          //if player empty handed PICK IT UP!          else if (!player.HasProp())          {get from cupboard to player hands              GetProp().SetPropParent(player);          }}} |

І ті скрипти які я згадував, але мало роскзазував. Саме PropSort і Props:

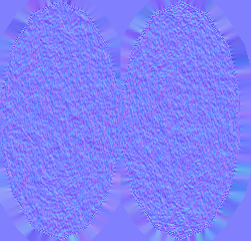
|  |  |
| --- | --- |
| [CreateAssetMenuAttribute()]  public class PropsSort : ScriptableObject  {      public Transform prefab;      public string propName;  } | PropsSort на початку містить атрибут [CreateAssetMenuAttribute()], що дозволяє створювати нові екземпляри цього класу |
| public class Props : MonoBehaviour  {      [SerializeField] private PropsSort propsSort;      private I\_PropParent propParent;      public PropsSort GetPropsSort()      {          return propsSort;      }      public void SetPropParent(I\_PropParent propParent)      {          // If this Props object already has a parent object,          // remove the reference to the Props object from that parent          if (this.propParent != null)          {              this.propParent.NullProp();          }          this.propParent = propParent;          propParent.SetPropParent(this);          //when changing parents we get new topPoint.          //Also make local coordinates 0 0 0          transform.parent = propParent.GetPropTransform();          transform.localPosition = Vector3.zero;      }      public I\_PropParent GetProp()      {          return propParent;      }  } | GetPropsSort(), повертає змінну propsSort.  SetPropParent(), приймає параметр propParent. У цьому методі перевіряється, чи існує вже батьківський об'єкт для екземпляру Props. Так - видаляє зв'язок між ним та попереднім батьківським об'єктом. Ні - новий батьківський об'єкт стає батьківським для Props.  Далі викликається метод SetPropParent() на propParent з параметром this, що передає власний екземпляр Props як child об'єкт до propParent. |

Повторно прикріплюю результат цього етапу:  
https://youtu.be/UzB2X9urgYA



Запікання текстур діло не дуже приємне

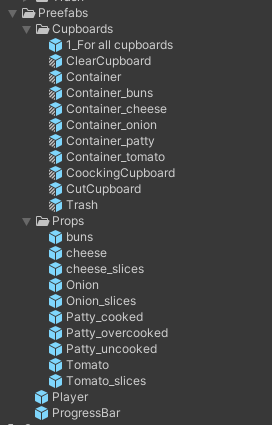


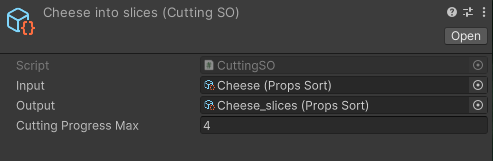


**Глава 6. «Let him cook»**

--Настав той час коли я описуватиму дуже мало. В основному я все пишу коментарями в коді. Дуже багато часу уходило, що я записував в цей док, тому зміна режиму.

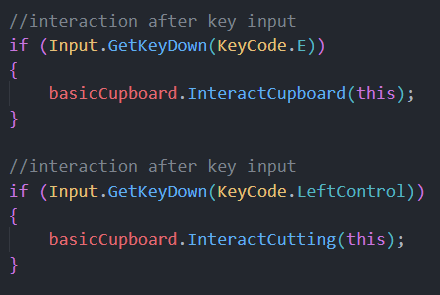
**Тут** буде йтись про додавання місця для **НАРІЗКИ** трьох об’єктів. **СМАЖЕННЯ** м’яса і ПЕРЕСМЕЖЕННЯ його. +установлені **PROGRESS BARS**. А і звичайно ж **СМІТНИК**

--тут дуже багато всього вже в сцені і пояснювати кожне – не варіант по часу.

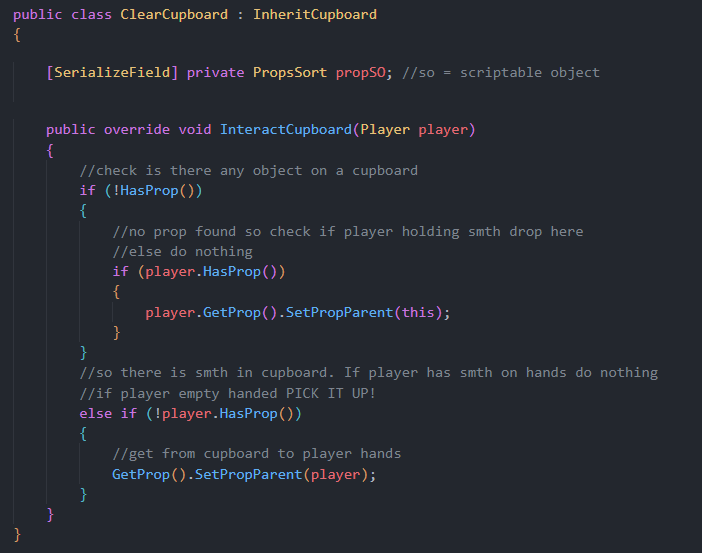
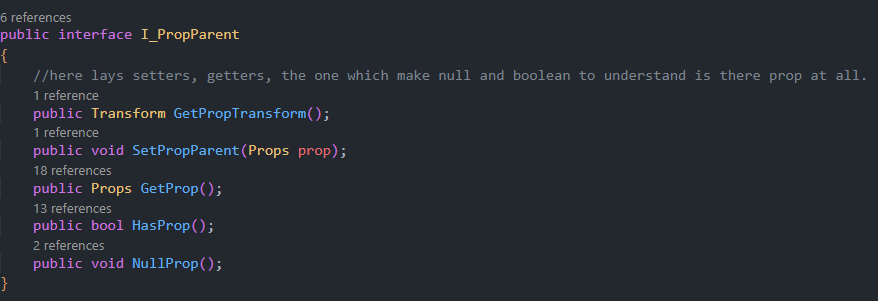


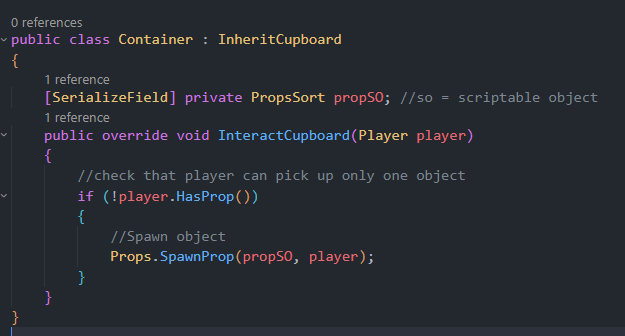
Можна помітити що ще з’явились prefabs з Котлетою(Patty) і Булками(Buns).

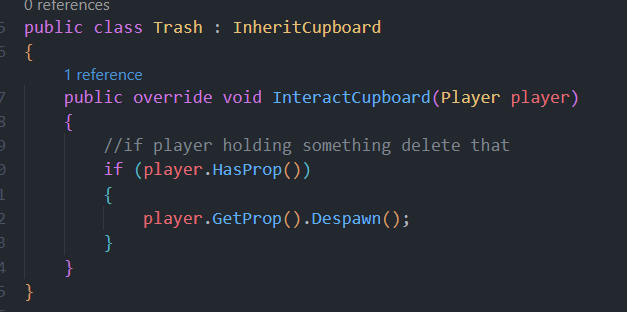
**Якнайкоротше** про наступні коди з точки зору механік. Рухатись wasd. Об’єкти з 5 контейнерів можна брати і ставити на звичайні clearCupboards (Ability 1). 3 інгредієнта можна нарізати на cuttingCupboard і інгредієнти стануть нарізаними їхніми версіями(нарізання клікання Ability 2). М’ясо можна покласти на пательню, через деякий час воно перейде у стан cooked і якщо м’ясо не прибрати вчасно то воно сгорить. І смітник, в який викидають непотрібні об’єкти.

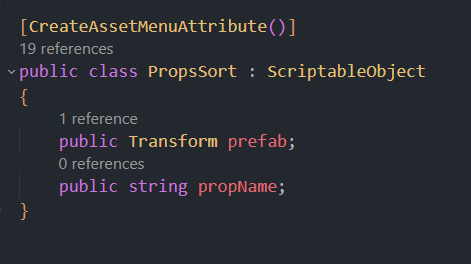
Тепер швидко по скріптах.

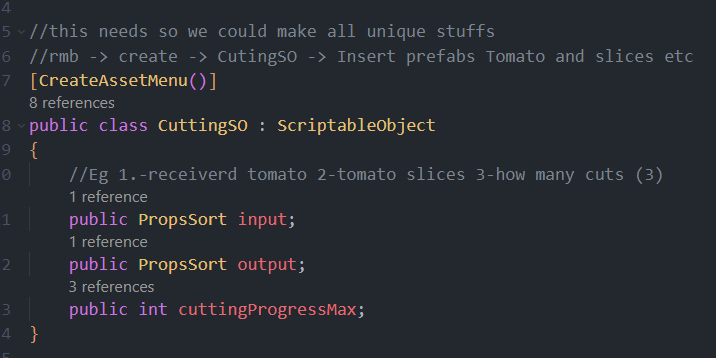
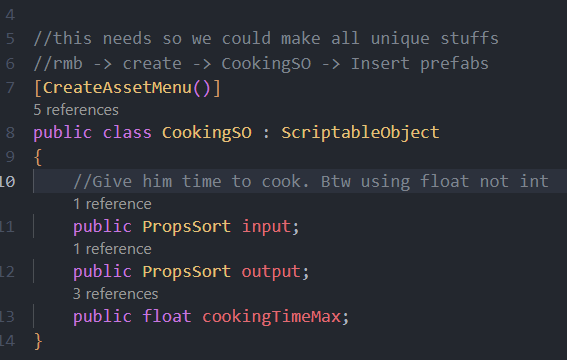
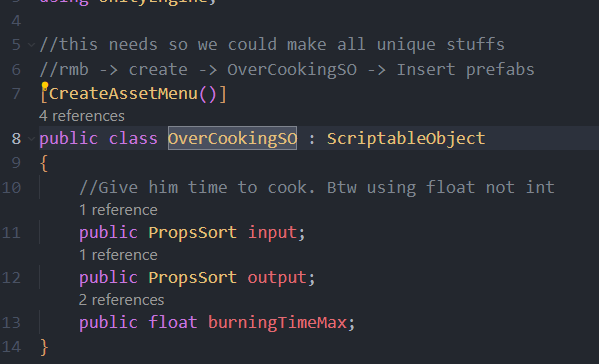
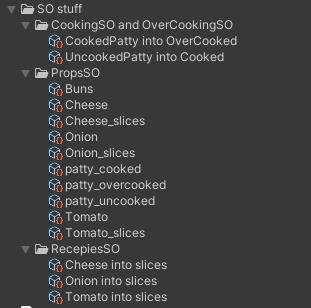
У Player додався момент з встановкою Ability2, але може потім зміню кнопки

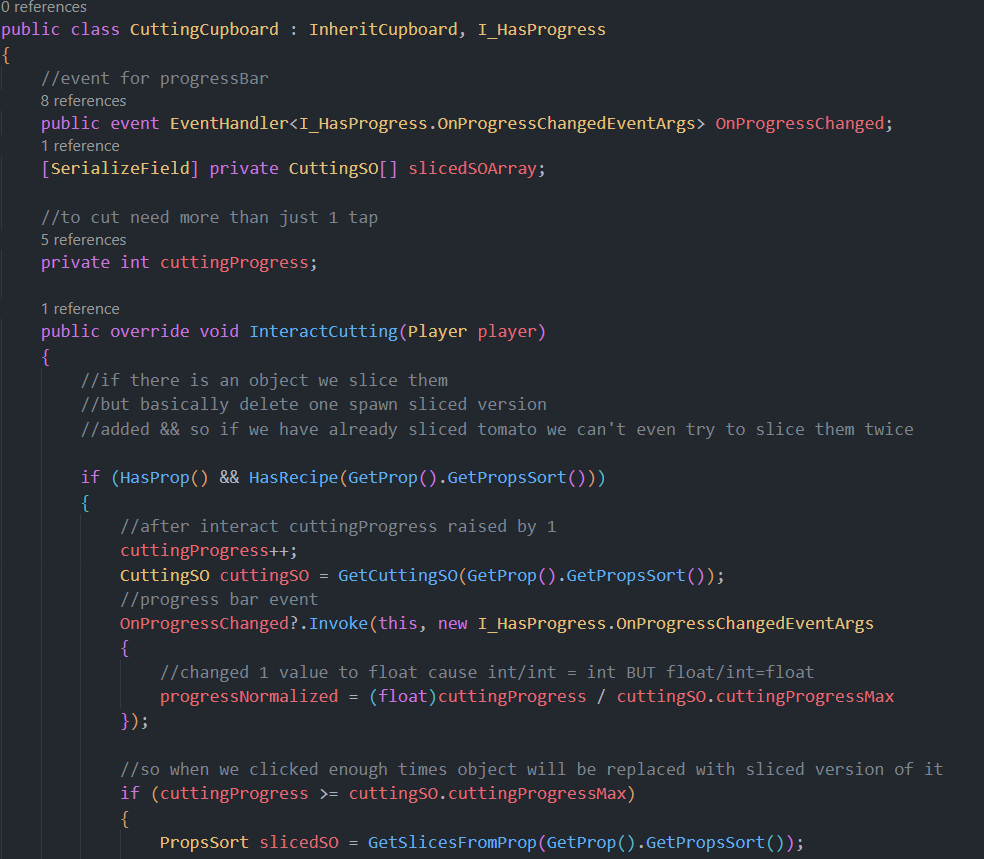
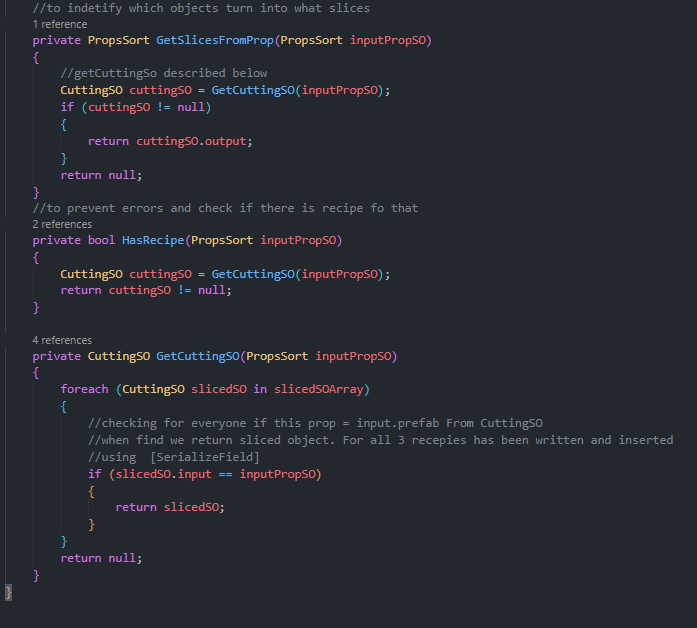
Тепер InheritCupboard і Interface PropParent. Всі пояснення в коментах. І останнє фото це простий clearCupboard.

Контейнер, який спавнить інгрідієнти:

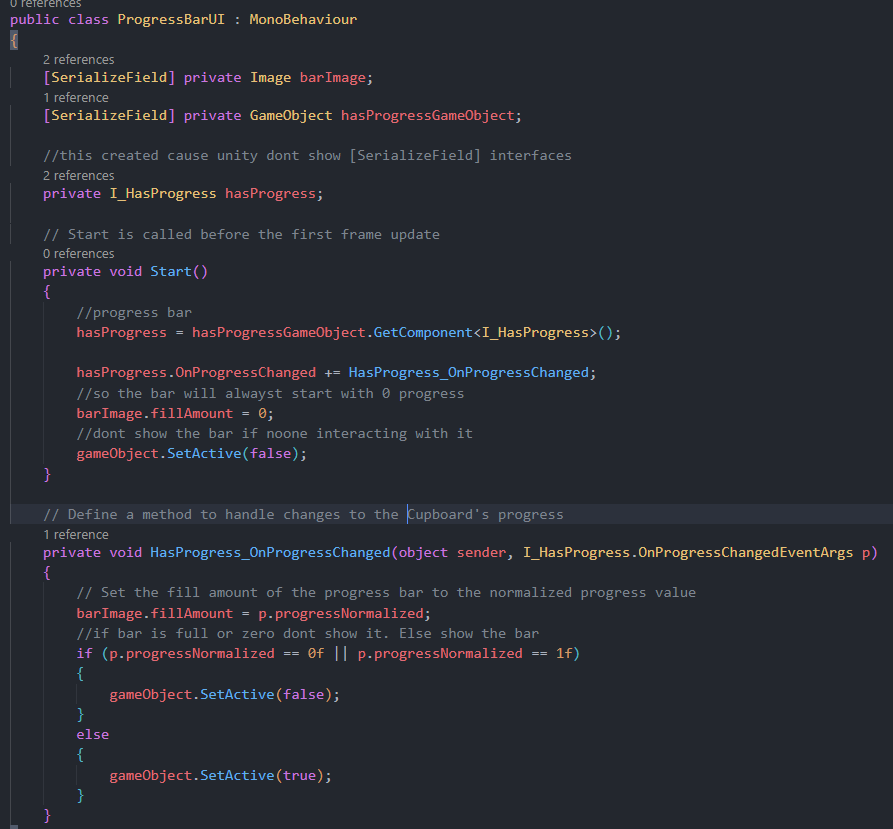
Смітник, який удаляє прийманий предмет

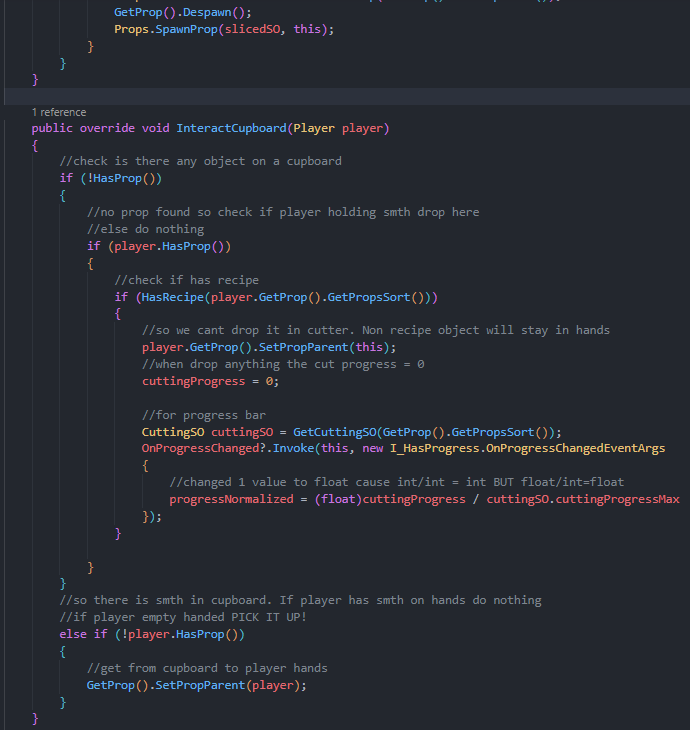
Скрипт об’єктів і SO (PropSort).

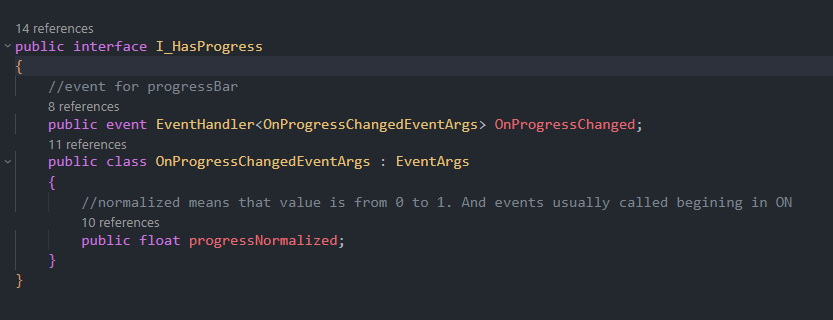
Зручне місце для показа всіх SO.

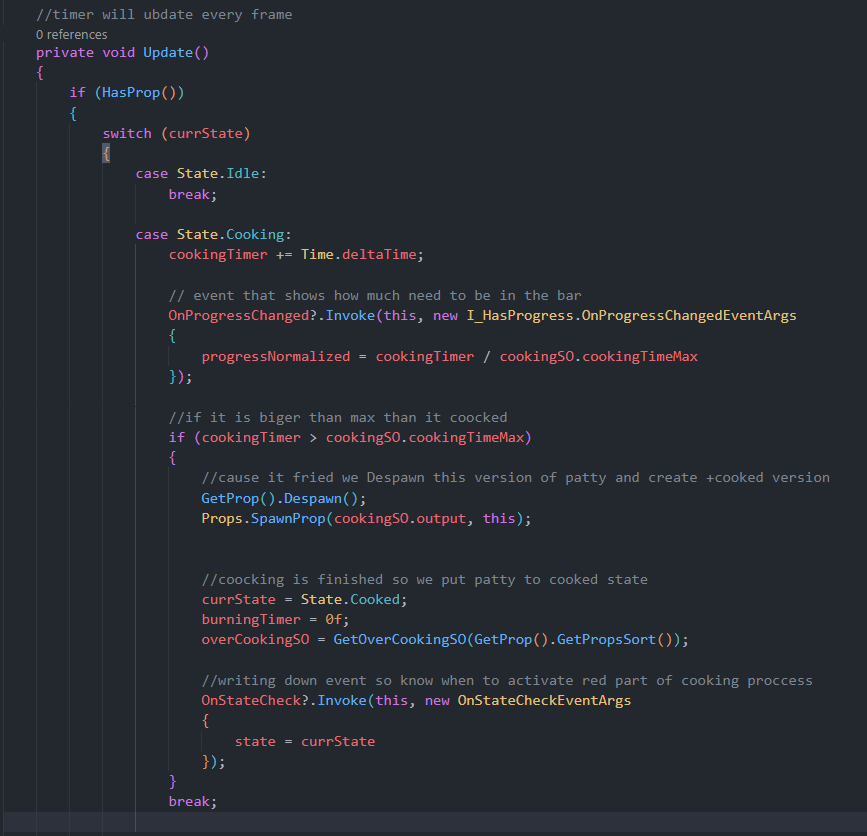
Нарізний стол:

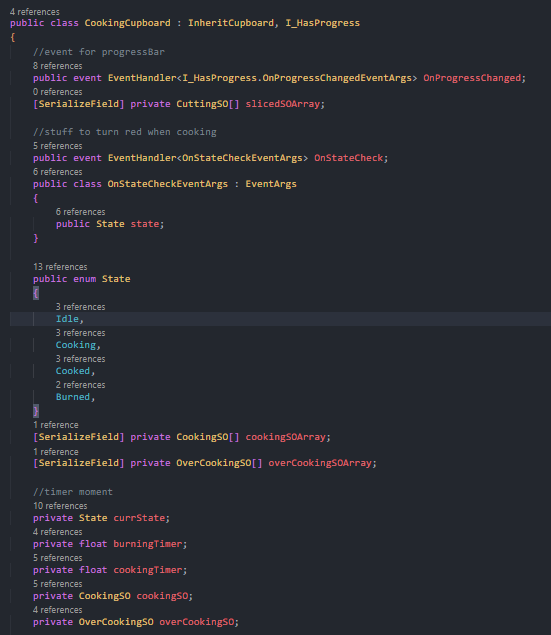
--праворуч 3 картинка.



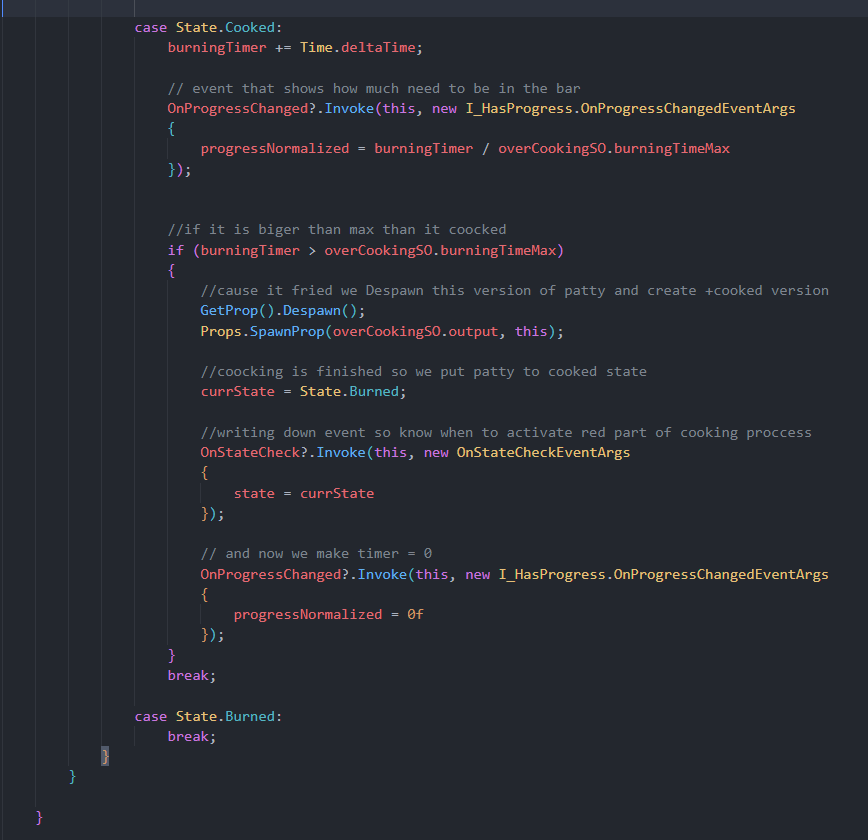
Праворуч наведена картинка про показник прогресу:

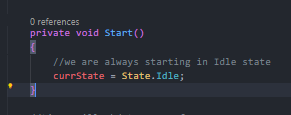
А це інтерфейс HasProgress

А тепер cookingCupboard, де ми використовуватимо State. Це створення станів, що відображають певний етап. Містить назви можливих станів. У нашому випадку використовується для відстеження процесу готування їжі.

* State.Idle: коли не використовується;
* State.Cooking: коли готується їжа;
* State.Cooked: коли їжа готова;
* State.Burned: коли згоріла їжа.

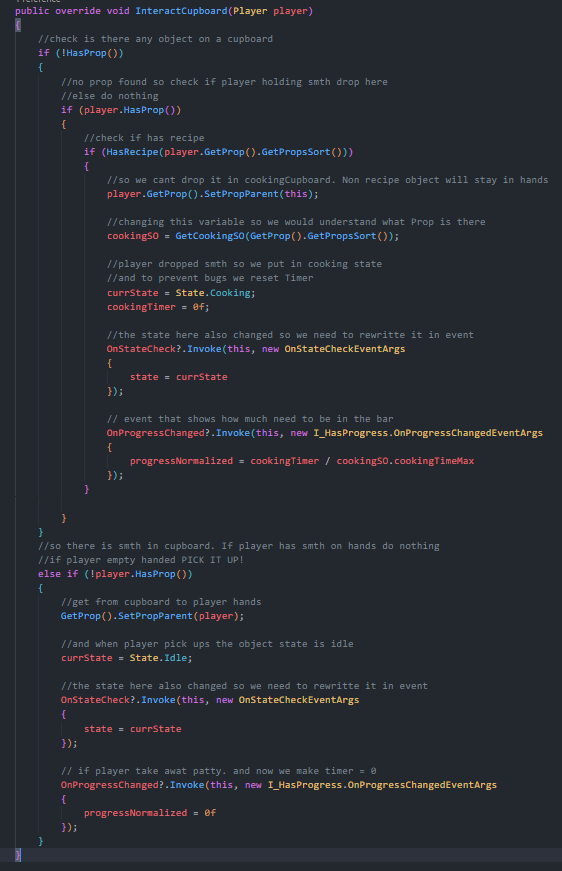
(тут про таймер і різні стани)

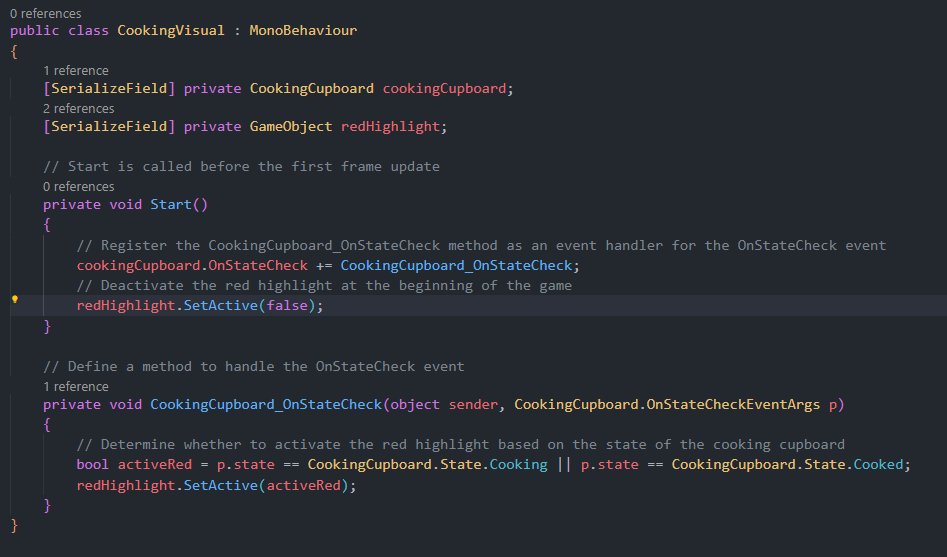




На наступній сторінці продовження -\_-

Якщо взаємодіять з ним



І тепер ще візуальна частина, що коли там готується щось то в нас плита становиться червоною. Але це ледве-ледве видно.

РЕЗУЛЬТАТ: https://youtu.be/zlKXuH4yras