**НОВОСИБИРСКАЯ АКАДЕМИЯ ДИЗАЙНА И ПРОГРАММИРОВАНИЯ**

Дипломная работа по специальности «Программирование»

«Создание пошаговой 2D стратегии на базе Unity»

Выполнил студент:

Маркидонов Владимир Владимирович

Новосибирск

2022г

ОГЛАВЛЕНИЕ

[1. ВВЕДЕНИЕ 3](#_Toc104603271)

[1.1. Выбор темы 3](#_Toc104603272)

[1.2. Теория 3](#_Toc104603273)

[1.2.1. Unity 3](#_Toc104603274)

[1.2.2. Microsoft Visual Studio 4](#_Toc104603275)

[1.2.3 Photoshop 5](#_Toc104603276)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 6](#_Toc104603277)

[2.1. Планирование 6](#_Toc104603278)

[2.2. Создание игры 7](#_Toc104603279)

[2.2.1. Передвижение 7](#_Toc104603280)

[2.2.2. Генератор уровня и фракции 11](#_Toc104603281)

[2.2.3. Движение камеры 17](#_Toc104603282)

[2.2.4. Зоны 18](#_Toc104603283)

[2.2.5. Оптимизация 21](#_Toc104603284)

[2.2.6. Магазин 22](#_Toc104603285)

[2.2.7. Юниты и здания 31](#_Toc104603286)

[2.2.8. Ходы и экономика 34](#_Toc104603287)

[2.2.8.1. Переход хода и очереди 34](#_Toc104603288)

[2.2.8.2. Экономика 38](#_Toc104603289)

[2.2.9. Переходы между сценами и их взаимодействие 41](#_Toc104603290)

[2.2.9.1. Главное меню 41](#_Toc104603291)

[2.2.9.2. Меню паузы 43](#_Toc104603292)

[2.2.10. Сохранения 45](#_Toc104603293)

[3. ЗАКЛЮЧЕНИЕ 52](#_Toc104603294)

[3.1. Баги и неполадки 52](#_Toc104603295)

[3.2. Вывод 52](#_Toc104603296)

[4. СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 53](#_Toc104603297)

# ВВЕДЕНИЕ

## Выбор темы

Я решил выбрать данную тему из-за давнего желания писать собственные игры. Играя в разные офлайн и онлайн игры гетерогенных жанров, порой мне хотелось добавить в них что-то от себя, я желал приложить руку к созданию крупных проектов и чтобы мой труд был оценён его пользователями. В НАДиП я познакомился с движками: Scratch, GameMaker, Kodu Game Lab, Unity. Трое из них я не решил рассматривать серьёзно для начала своей деятельности разработки ибо не увидел в них потенциала создавать абсолютно любой свой плод воображения, потому и остановился на Unity – сложном и непонятном.

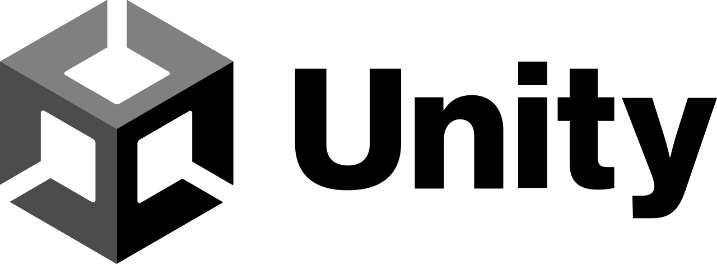
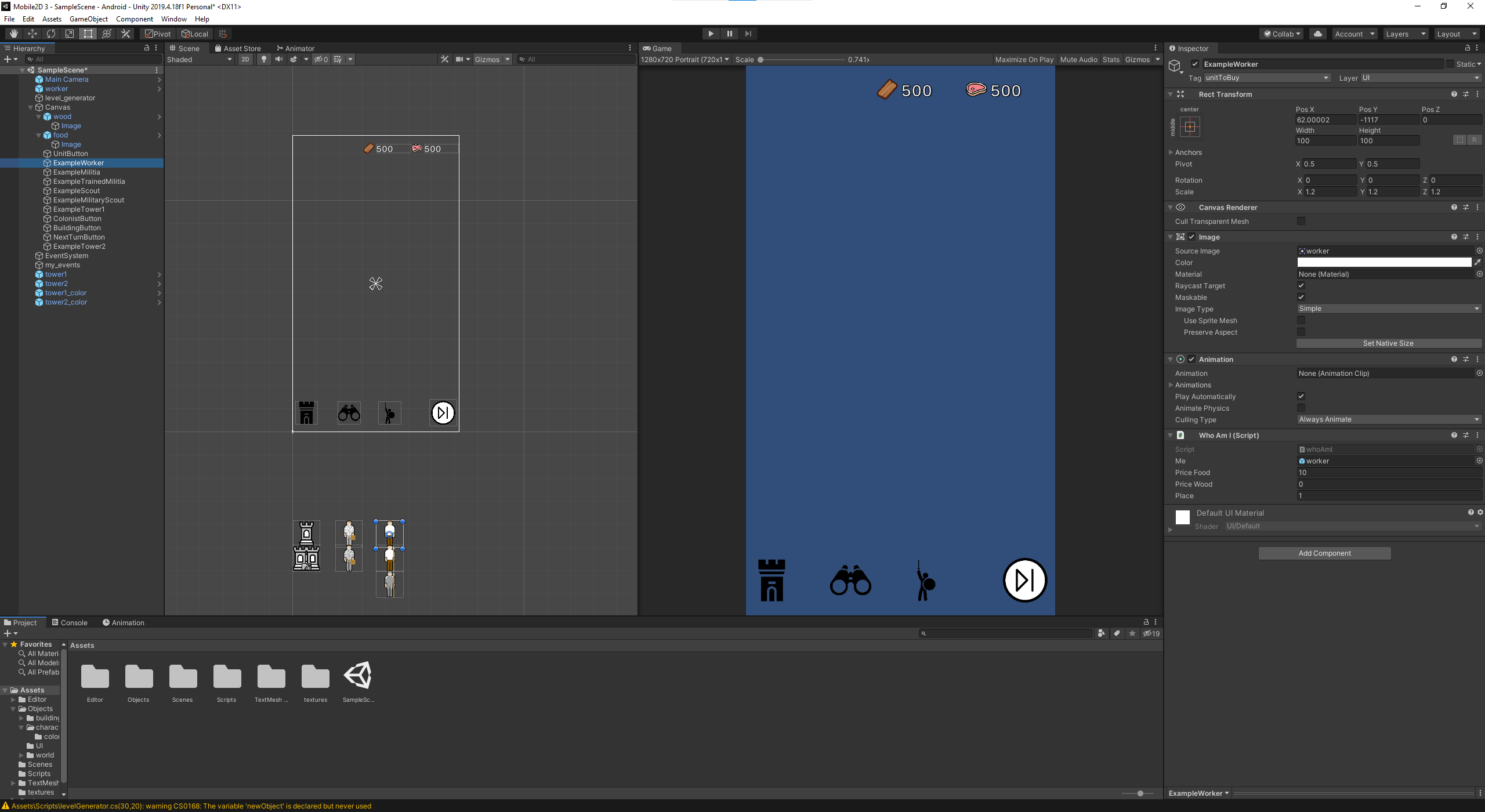
Я рассматривал и другие варианты движков, например Unreal Engine, но решил всё-таки остановиться на Unity, так как не было желания обучаться с нуля, ведь на нём уже была база, полученная с уроков НАДиП.

Мой выбор создания 2D пошаговой стратегии, связан с популярностью игры «Antiyoy» в среде моего класса. Её геймплей был взят за модель для разработки. Также более по этой причине платформой для игры стала android.

## Теория

### Unity

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Среди них: Windows, OS X, Windows Phone, Android, Apple iOS, Linux, Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, MotionParallax3D. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

*\*Логотип и интерфейс Unity*

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

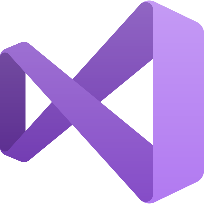
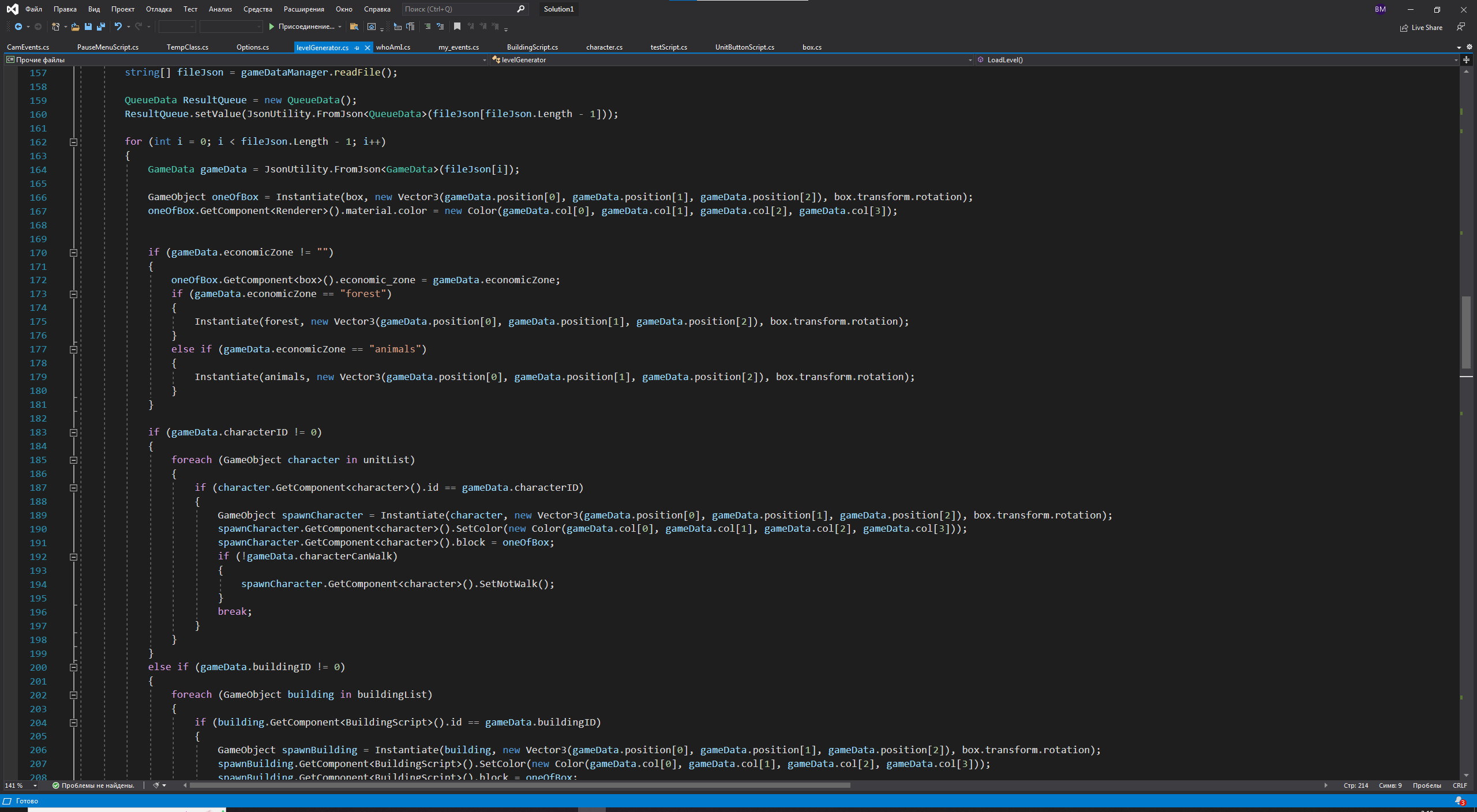
На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями.

Проект в Unity делится на сцены (уровни) — отдельные файлы, содержащие свои игровые миры со своим набором объектов. Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты.

### Microsoft Visual Studio

Так как я имел небольшой опыт работы с языками Си именно в Visual Studio, я решил выбрать именно его в качестве среды разработки скриптов.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в машинном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ.

*\*Логотип и интерфейс Microsoft Visual Studio*

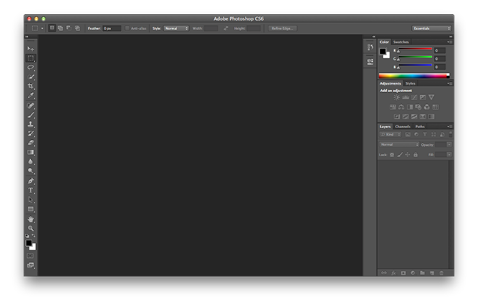
Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

1. Редактор поддерживает языки: C#, Razor, Typescript, Visual Basic, JavaScript, CSS, Less, Sass и Font Awesome, Python, C++, F#

### Photoshop

Я имею давний опыт работы с фотошопом. Именно его я решил использовать для рисования собственных текстур. Но не все текстуры, представленные в игре являются моими, однако большинство это всё же мои.

Adobe Photoshop — многофункциональный графический редактор, разрабатываемый и распространяемый компанией Adobe Systems. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известной программой разработчика.



*\*Логотип и интерфейс программы*

В настоящее время Photoshop доступен на платформах macOS, Windows и iPadOS. Для Windows Phone и Android доступна упрощённая версия приложения под названием Adobe Photoshop Touch. Также существует версия Photoshop Express для Windows Phone 8 и 8.1. В 2014 году в США проходило бета-тестирование потоковой версии продукта для Chrome OS. Ранние версии редактора были портированы под SGI IRIX, но официальная поддержка прекращена начиная с третьей версии продукта. Для версий 8.0 и CS6 возможен запуск под Linux с помощью альтернативы Windows API — Wine.

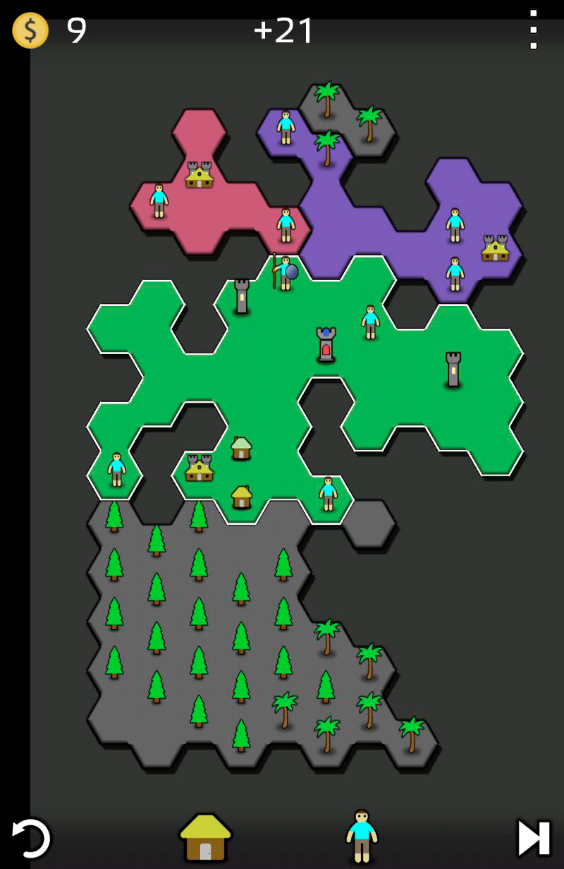
Хотя изначально программа была разработана как редактор изображений для полиграфии, в данное время она широко используется и в веб-дизайне.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Планирование

С самого начала у меня был какой-то план и я его придерживаться. Главная философия, которую я постановил для себя – не стараться сделать код идеальным. Поиск лучшего варианта занимает очень много времени и сил из-за моей неопытности, поэтому было решено использовать то, что понятно, а после корректировать, если найдётся альтернатива. Из-за этого основные скрипты геймплея получились плохо оптимизированы для игры на больших уровнях, но об этом позже.

И так. Рассмотрим модель «Antiyoy»:

*\*Скриншоты из игры «Antiyoy»*

Игра состоит из пяти общих составляющих: карта, передвижение, фракции, экономика, магазин. Задача была расставить эти фрагменты в порядке от более независимых к зависимым от других, а также так, чтобы не потерять интерес к проекту, во время его написания (это стало важным из-за тяжёлого старта разработки игры: я упоминал, что у меня была база, но это лишь представление, как должно выглядеть, но не как это написать)

Я хотел начать с двух вещей: передвижение и генерация уровня. Именно это я посчитал фундаментом для всего остального. Передвижение стало первым шагом в плане, по собственным предрассудкам, а значит вторым - генерация. После план менялся и итогом стала последовательность: Передвижение > Генерация карты > Магазин > Фракции > Экономика. Все они были реализованы. Если рассматривать итоговый план разработки с большими деталями, то получится содержание данного документа.

## Создание игры

Это был крайне трудный процесс для меня по началу, но чем дальше я шёл, тем легче мне было.

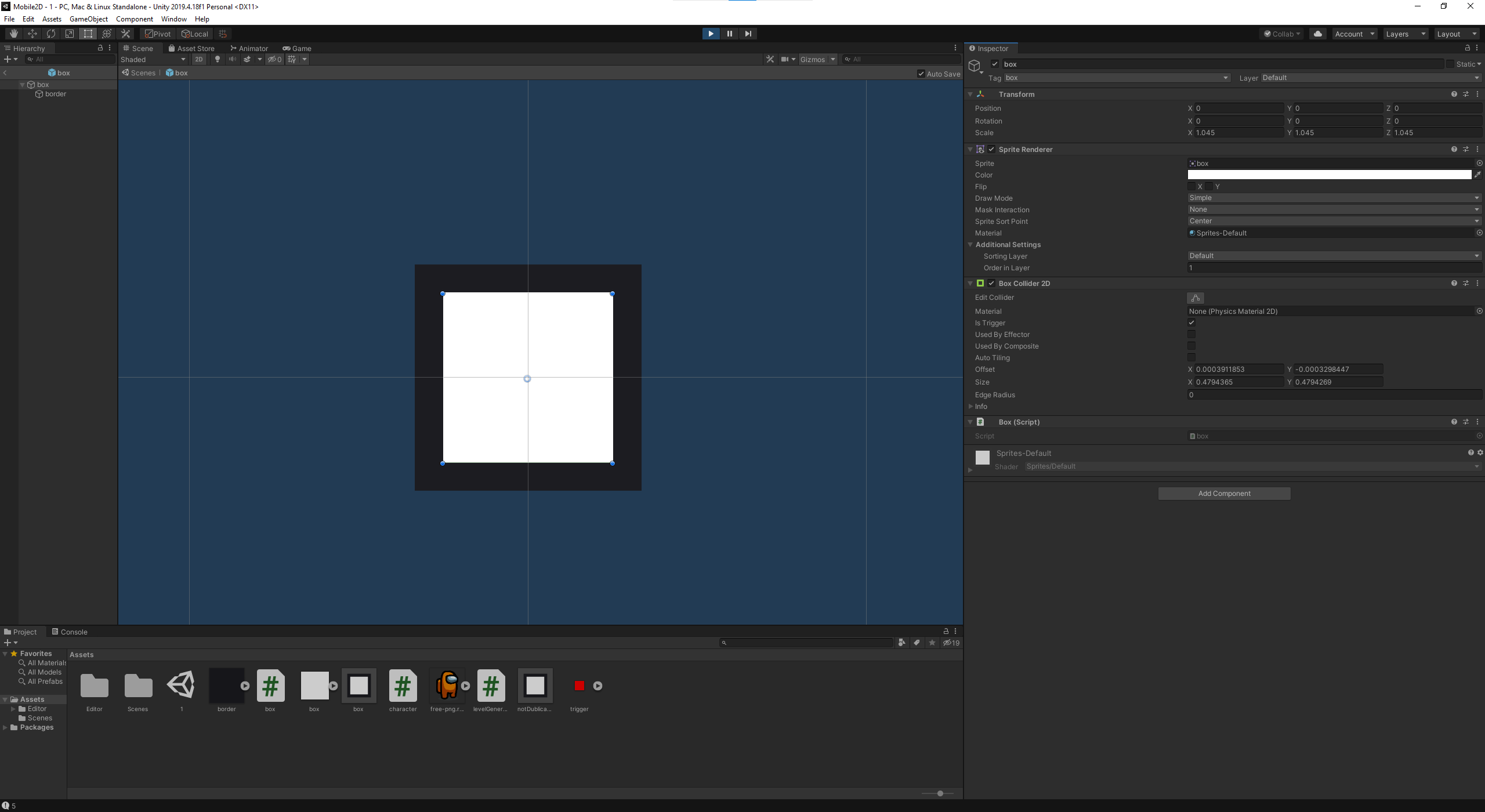
### Передвижение

Несмотря на простоту начального кода, до него я дошёл через день размышлений.

И так. Вспоминая игру-модель, выделяется 3 основных составляющих движения:

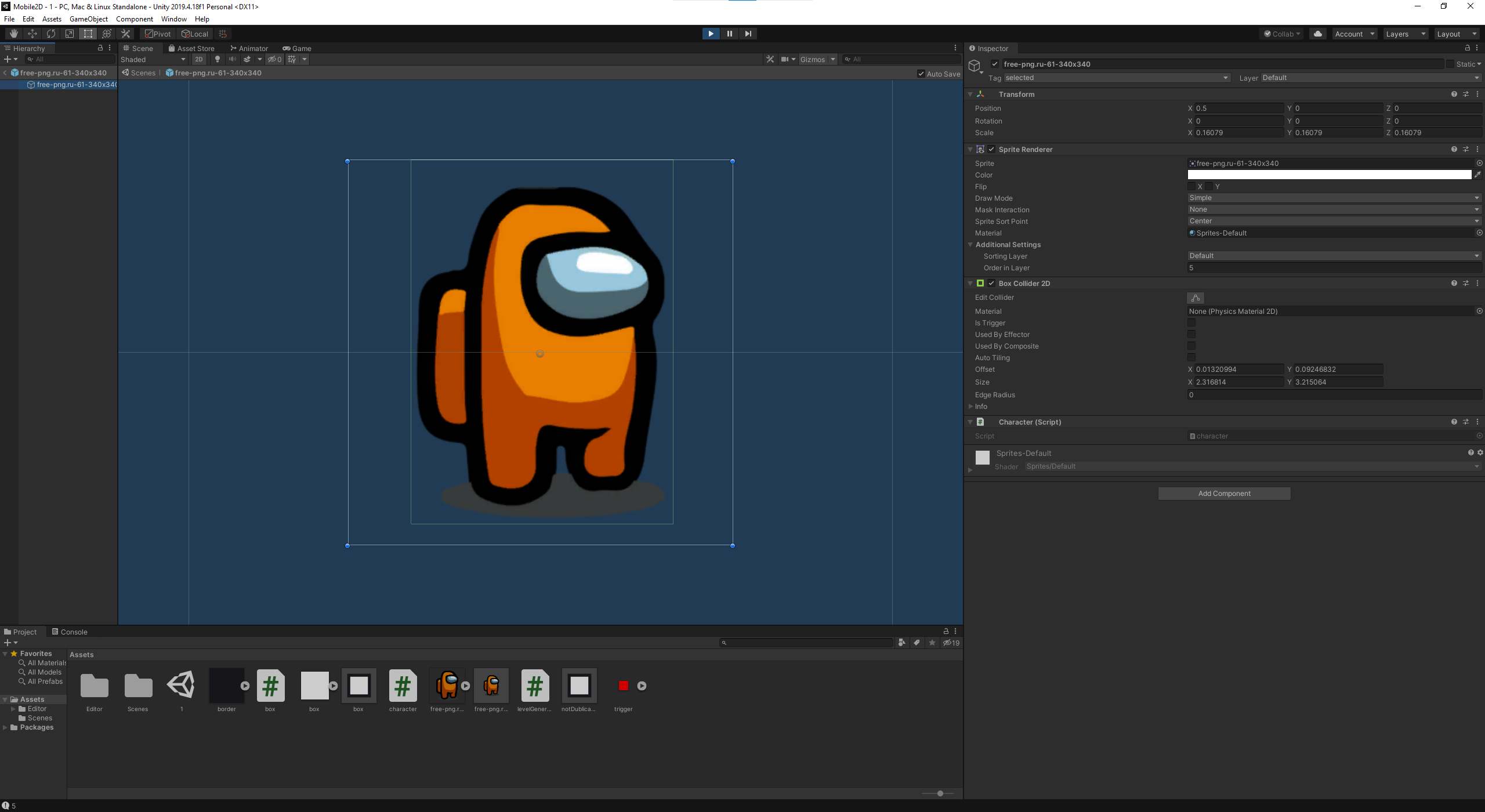
1. Выделение персонажа при нажатии на него
2. Перемещение выделенного персонажа нажатием на блок, находящийся в зоне досягаемости персонажа
3. Снятие выделения (отмена хода) персонажа нажатием за недосягаемую зону.

Для начала игры, должна быть территория, ведь на ней завязана вся игра (персонажи, здания, деревья). Совместив две текстуры и подобрав правильный размер, я создал объект – коробку (блок):



Данный префаб состоит из двух частей: границы и самой коробки. Граница имеет декоративную роль, а коробка – основную, поэтому к ней привязана граница, а не наоборот.

Далее необходимо было создать любой объект, который бы был прототипом персонажа. Он выглядел так (это только прототип персонажа, поэтому у его странное название и текстура из другой случайной игры):



Осталось только написать скрипт для них.

Скрипт для коробки:

// Стандартный импорт библиотек Unity

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class box : MonoBehaviour // Класс представляющий объект блока

{

public void OnMouseDown() // Срабатывание при нажатии на блок

{

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected"); // Найти все объекты с тэгом "selected" и записать в массив

selected[0].transform.position = transform.position; // Первому в списке выделенных присвоить позицию блока

selected[0].tag = "character"; // Установить тэг "character"

}

}

Скрипт для персонажа:

// Стандартный импорт библиотек Unity

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class character : MonoBehaviour // Класс представляющий объект персонажа

{

public void OnMouseDown() // Срабатывание при нажатии на персонажа

{

gameObject.tag = "selected"; // Установить тэг "selected"

}

}

Как видно для выделения персонажа, необходимо нажать по нему и присвоить тэг “selected” (выделен), а чтобы передвинуть его – нажать любой блок, после чего тэг выделения смениться на “character” – просто персонаж.

Создадим уровень из пары блоков и персонажа и проверим как работают скрипты:

1.  2) 

*\*1 Начальное положение и выделение персонажа*

*\*2 Перемещение выделенного персонажа*

Наш игровой объект ожил, но ходит он на неограниченном расстоянии, а также этот способ не лишён багов. Например герой может перестать выделяться ступая на некий “блок ловушку” и ещё можно выделить больше двух персонажей. Стоит это исправить:

Блок:

// Новая переменная GameObject character

void OnMouseDown()

{

//Debug.Log(character);

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected");

// Если есть веделенный персонаж и у блока нет своего персонажа, то передвинуть выделенного на себя и снять выделение

if ((selected.Length == 1) & (character == null))

{

if (Vector3.Distance(selected[0].transform.position, gameObject.transform.position) <= 2.5f)

{

selected[0].transform.position = transform.position;

selected[0].tag = "character";

character = selected[0];

}

}

// Если блок имеет в себе персонажа

else if (character != null)

{

// И он уже не на блоке

if (character.transform.position != transform.position)

{

// То если есть выделенный - передвинуть его на себя при соблюдении расстояния, а старого отвязать

if (Vector3.Distance(selected[0].transform.position, gameObject.transform.position) <= 2.5f)

{

character.tag = "character";

if (selected.Length == 1)

{

character = selected[0];

selected[0].transform.position = transform.position;

selected[0].tag = "character";

GameObject[] my\_events = GameObject.FindGameObjectsWithTag("my\_events");

my\_events[0].GetComponent<my\_events>().RemoveZone();

}

}

}

// Если блок имеет в себе персонажа и нет выделенных, то выделить своего персонажа и отвязать его от себя

else if (selected.Length == 0)

{

SeeZone();

character.tag = "selected";

character = null;

}

// Если блок имеет в себе персонажа и есть выделенный, то убрать выделение

else if (selected.Length == 1)

{

selected[0].tag = "character";

}

}

}

У блока появилась привязка к персонажу – переменная “character”. Благодаря этому исправляется баг с “блоками-ловушками”.

Персонаж:

public GameObject block;

public void OnMouseDown()

{

// Если есть выделенный - отметить выделение, иначе выделить самого себя

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected");

if (selected.Length == 1)

{

GameObject[] my\_events = GameObject.FindGameObjectsWithTag("my\_events");

my\_events[0].GetComponent<my\_events>().RemoveZone();

selected[0].tag = "character";

}

else

{

gameObject.tag = "selected";

block.GetComponent<box>().SeeZone();

}

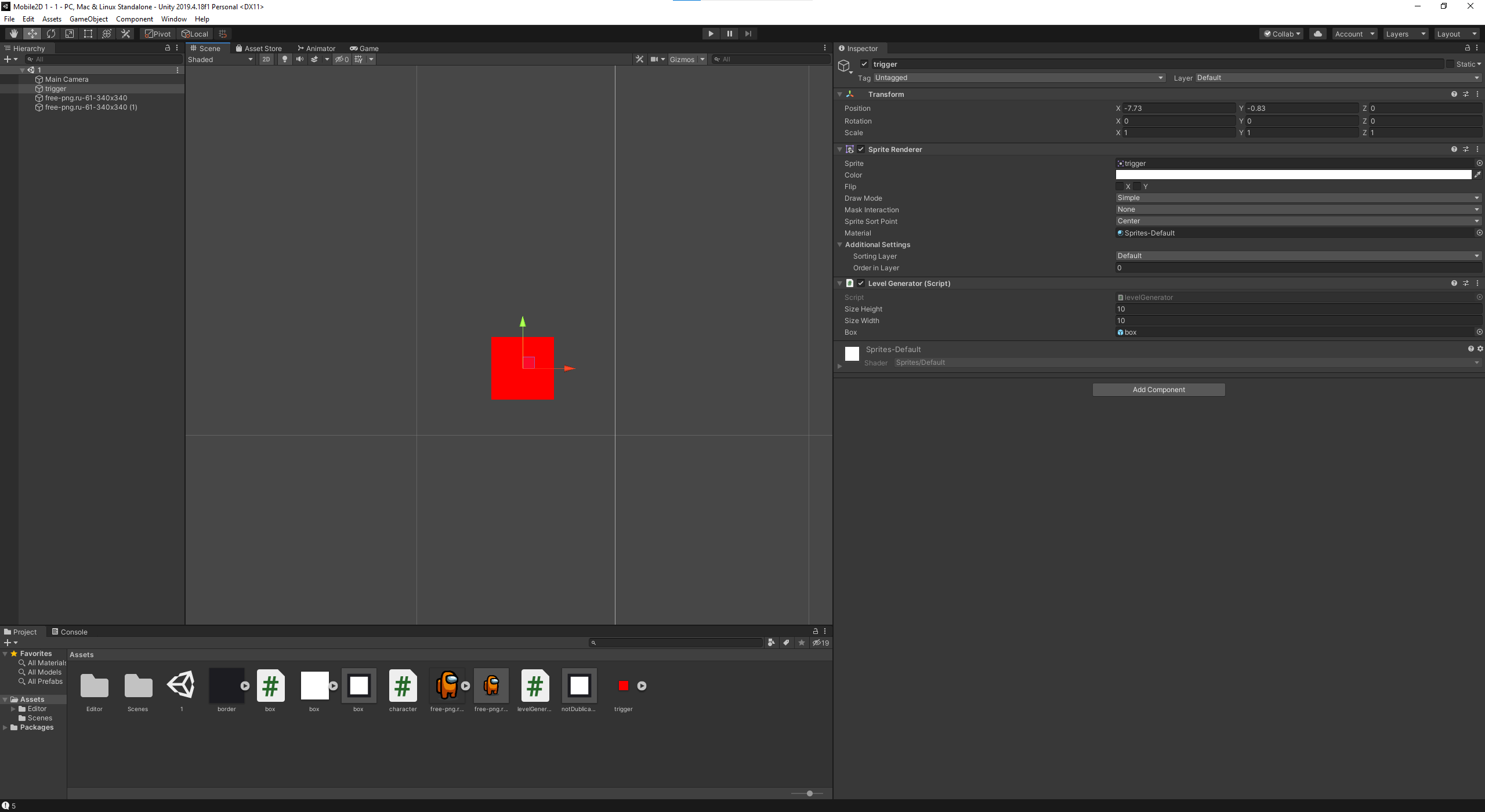
}

У персонажа появилась привязка к блоку – переменная “block”, но пока она не используется. В общем тут особо ничего не изменилось.

Как можно заметить, в коде появились функции “SeeZone” и “RemoveZone”, а также есть обращения к объекту с тэгом “my\_events”, он имеет схожее название с тэгом. Пока это нововведения не играют никакой роли. Я расскажу о них, когда они будут работать.

### Генератор уровня и фракции

Теперь стоит добавить и генератор уровня:



Это будет красный квадрат с именем “trigger”. На него наложим следующий скрипт:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Random = System.Random; // Импорт библиотеки случайных чисел

public class levelGenerator : MonoBehaviour

{

// Start is called before the first frame update

Random rnd = new Random(); // Сгенерировать случайное число

float distance = 0.5f; // Дистанция между блоками

public float sizeHeight = 10.0f; // Высота уровня в блоках

public float sizeWidth = 10.0f; // Ширина уровня в блоках

public GameObject box; // Объект блока

int place; // Вспомогательная переменная

void CreateObject(Vector3 pos) // Функция создания блока

{

Instantiate(box, pos, transform.rotation);

}

void Start() // Функция генерации мира. Запускается с загрузкой сцены

{

// Преобразования размеров уровня в блоках, в размеры в координатах

float sizeWidthTemp = sizeWidth \* distance;

Debug.Log(sizeWidthTemp);

float sizeHeightTemp = sizeHeight \* distance;

Debug.Log(sizeHeightTemp);

while (sizeHeightTemp != 0 - distance)

{ // Циклы для пробега по всем возможным координатам блоков для уровня

while (sizeWidthTemp != 0 - distance)

{

place = rnd.Next(0, 2); // Генерация случайного числа от 0 до 1

if (place == 1)

{

CreateObject(new Vector3(sizeWidthTemp, sizeHeightTemp)); // Создать блок

}

sizeWidthTemp -= distance; // Сдвиг влево

}

sizeHeightTemp -= distance; // Сдвиг вниз

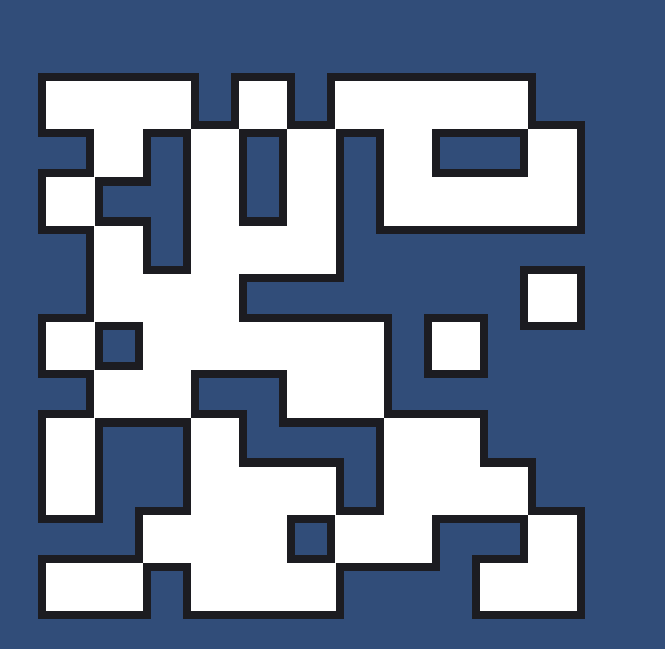
sizeWidthTemp = sizeWidth \* distance; // Сдвиг в правый край

}

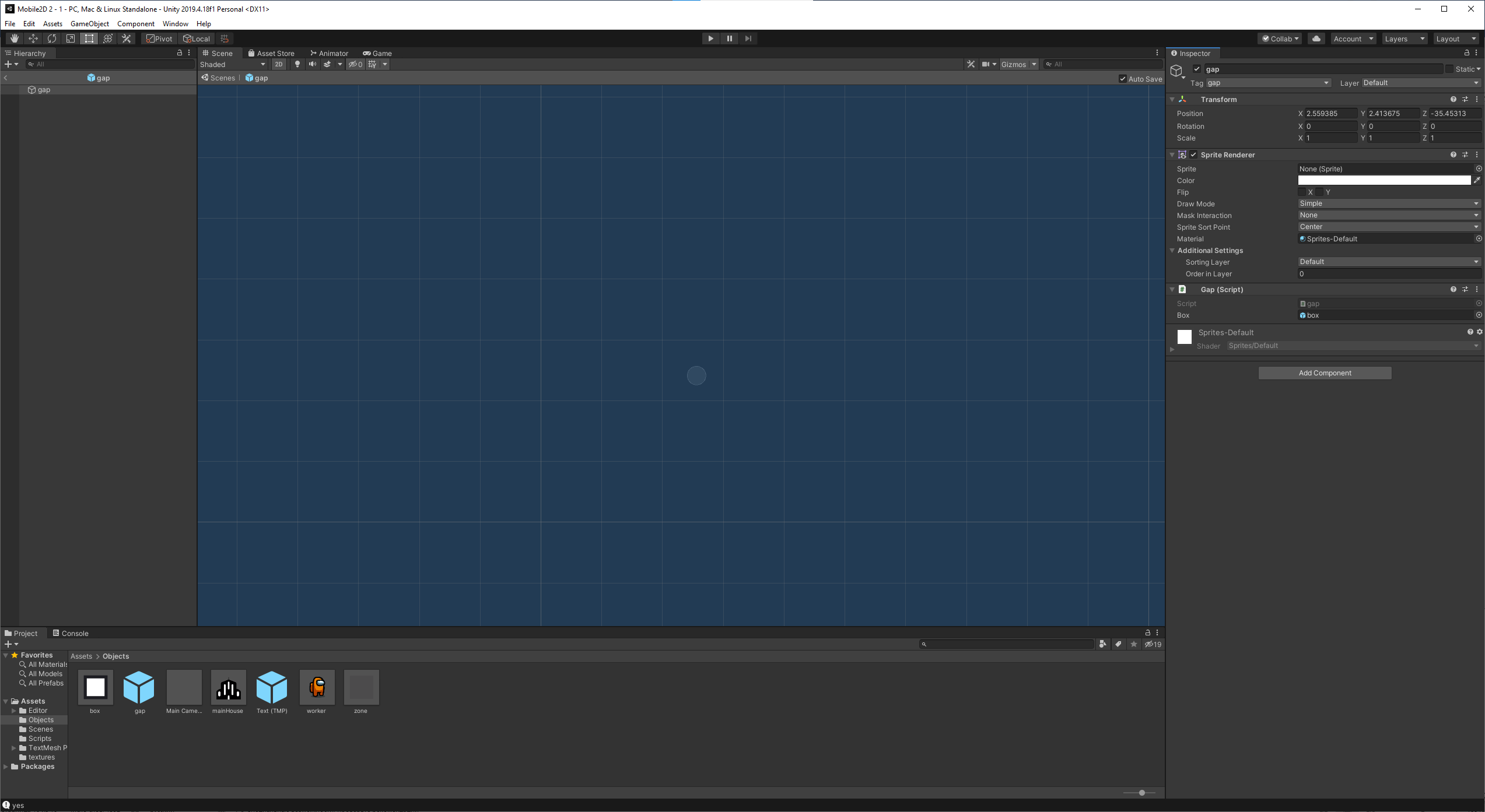
}

}

Итог работы скрипта:



Ну это слишком просто. Стоит усложнить алгоритм. Но для этого нам понадобится новый вспомогательный префаб “gap” – пустота:



Да, это пустота. Она будет использоваться для корректировки вида уровня. Для неё необходим скрипт (стандартный импорт присутствует, просто я его не указываю):

public class gap : MonoBehaviour

{

public GameObject box;

public void Correct(float distance) // Функция вызываемая генератором уровня

{

float spawn = 0; // Количество блоков, находящихся в дистанции, передаваемой генератором

// Аглоритм находит все блоки в указанной дистанции от объекта

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

for (int i = 0; i < boxes.Length; i++)

{

if (Vector3.Distance(boxes[i].transform.position, transform.position) == distance)

{

spawn++;

if (spawn >= 3) // Если количество блоков равно 3, создать коробку и завершить цикл

{

Instantiate(box, transform.position, transform.rotation);

break;

}

}

}

gameObject.tag = "Destroyed";

Destroy(gameObject); // В конце всегда разрущение пустоты

}

}

Рассмотрим новую функцию в коробках:

public GameObject zone;

GameObject character;

GameObject building;

List<GameObject> zoneBlocks = new List<GameObject>();

public void Correct(float distance) // Аналогично с gap, проверяет есть ли начилие блоков рядом с ним

{

int match = 0;

bool alive = false;

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

for (int i = 0; i < boxes.Length; i++)

{

if (Vector3.Distance(boxes[i].transform.position, transform.position) == distance)

{

match += 1;

if (match == 1) // Если рядом с объектом есть один блок, то он останется на сцене, иначе разрушится

{

alive = true;

break;

}

}

}

if (!alive)

{

gameObject.tag = "Destroyed";

Destroy(gameObject);

}

}

На этих нововведениях основывается обновлённый генератор:

using System.Collections;

using System;

using System.Collections.Generic;

using UnityEngine;

using Random = System.Random;

public class levelGenerator : MonoBehaviour

{

// Start is called before the first frame update

Random rnd = new Random();

float distance = 0.5f;

public int factionsCount = 2; // Количество фракций на уровне

public float sizeHeight = 10.0f;

public float sizeWidth = 10.0f;

public GameObject box;

public GameObject gap;

public GameObject mainHouse; // Главное здание фракции

public GameObject camera;

int randInt; // Всмомогательная переменная

public bool camSet = true; // Отвечает за установку камеры по центру уровня

void CreateObject(Vector3 pos)

{

randInt = rnd.Next(0, 2);

if (randInt == 1)

{

Instantiate(box, pos, transform.rotation);

}

else

{

Instantiate(gap, pos, transform.rotation);

}

}

void Start()

{

float sizeWidthTemp = sizeWidth \* distance;

float sizeHeightTemp = sizeHeight \* distance;

// Установка камеры по центру уровня

float max = Math.Max(sizeWidth, sizeHeight);

camera.transform.position = new Vector3(sizeWidthTemp / 2, sizeHeightTemp / 2, -10);

if (camSet) {

camera.GetComponent<Camera>().orthographicSize = max \* (0.33f - (max / 10 \* 0.006f));

}

// Начальная генерация

while (sizeHeightTemp != 0 - distance)

{

while (sizeWidthTemp != 0 - distance)

{

CreateObject(new Vector3(sizeWidthTemp, sizeHeightTemp, 0));

sizeWidthTemp -= distance;

}

sizeHeightTemp -= distance;

sizeWidthTemp = sizeWidth \* distance;

}

// Первое преобразование блоками

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

for (int i = 0; i < boxes.Length; i++)

{

boxes[i].GetComponent<box>().Correct(distance);

}

// Второе преобразование пустотами

GameObject[] gaps = GameObject.FindGameObjectsWithTag("gap");

for (int i = 0; i < gaps.Length; i++)

{

gaps[i].GetComponent<gap>().Correct(distance);

}

// Третье преобразование блоками

boxes = GameObject.FindGameObjectsWithTag("box");

for (int i = 0; i < boxes.Length; i++)

{

if (boxes[i].tag == "box")

{

boxes[i].GetComponent<box>().Correct(distance);

}

}

// Размещение главных домов фракции и раскраска блоков под ними

boxes = GameObject.FindGameObjectsWithTag("box");

Color prevColor = new Color(1, 1, 1);

int prevRand = -1;

for (int i=0; i<factionsCount; i++)

{

randInt = rnd.Next(0, boxes.Length);

if (boxes[randInt] == null)

{

Debug.Log("no");

} else { Debug.Log("yes"); }

Color newColor = new Color(rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f);

if ((newColor == new Color(1,1,1)) || (newColor == prevColor) || (randInt == prevRand))

{

i--;

}

else

{

boxes[randInt].GetComponent<Renderer>().material.color = new Color(rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f);

prevColor = boxes[randInt].GetComponent<Renderer>().material.color;

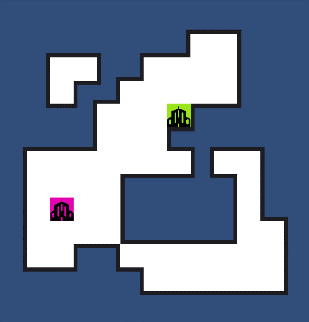
GameObject house = Instantiate(mainHouse, boxes[randInt].transform.position, boxes[randInt].transform.rotation);

house.GetComponent<Renderer>().material.color = prevColor;

}

}

}

}

Знаменательный момент, в генераторе также появляются фракции! Сейчас они представляют из себя лишь зачатки будущей системы. Далее я буду развивать её. Это было достаточно сложно

*\*Пример работы генератора*

Механика фракций реализована следующим образом. В начале есть генератор, который создаёт основные точки с главными зданиями и перекрашивает ближайшие 8 блоков от главного здания в цвет фракции. Чтобы фракции не накладывались друг на друга, существует минимальное расстояние, которое должно соблюдаться между главными зданиями. За это отвечает данный участок скрипта генератора уровня:

boxes = GameObject.FindGameObjectsWithTag("box");

GameObject[] factions = new GameObject[factionsCount]; // Массив блоков с главными зданиями

int toMany = 0; // Количество неудачных попыток выбора блока с главным зданием

List<GameObject> prohibitedBlocks = new List<GameObject>(); // Список блоков, которые были задейстованны в неудачных попытках

// Определение главных блоков фракций

for (int i=0; i<factionsCount; i++)

{

randInt = rnd.Next(0, boxes.Length); // Случайный индекс блока в массиве boxes

newColor = new Color(rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f, rnd.Next(0, 11) / 10f); // Случайный цвет

bool allNotGood = false; // Переменная ответственная за пропуск дальнейших действий цикла

// Проверка на возможность установки главного здания фракции на этом блоке, иначе цикл задерживается до ограничения ошибок или пока не будет найден идеальный блок

foreach (GameObject factionBlock in factions)

{

Debug.Log(factionBlock);

if (factionBlock == null)

{

break;

}

else if (prohibitedBlocks.Contains(boxes[randInt]) )

{

// Задержка цикла

allNotGood = true;

i--;

toMany++;

break;

}

else if ( ( (toMany<10) | (boxes[randInt] == factionBlock) ) & ( (newColor == new Color(1, 1, 1)) | (newColor == factionBlock.GetComponent<Renderer>().material.color) | (boxes[randInt].transform.position == factionBlock.transform.position) | (Vector3.Distance(boxes[randInt].transform.position, factionBlock.transform.position) < 2) ) )

{

Debug.Log(newColor);

Debug.Log(boxes[i]);

Debug.Log(Vector3.Distance(boxes[randInt].transform.position, factionBlock.transform.position));

Debug.Log(factionBlock.GetComponent<Renderer>().material.color);

Debug.Log(factionBlock);

// Задержка цикла

allNotGood = true;

i--;

toMany++;

prohibitedBlocks.Add(boxes[randInt]);

break;

}

}

if (allNotGood)

{

continue;

}

else

{

factions[i] = boxes[randInt]; // Записть блока в массив блоков с главными зданиями

prohibitedBlocks.Clear();

}

// Преобразование блока в главный блок фракции

boxes[randInt].GetComponent<Renderer>().material.color = newColor;

GameObject house = Instantiate(mainHouse, boxes[randInt].transform.position, boxes[randInt].transform.rotation);

boxes[randInt].GetComponent<box>().building = house;

boxes[randInt].GetComponent<box>().power = 1;

house.GetComponent<Renderer>().material.color = newColor;

toMany = 0; // Сброс попыток

int blocks = 1; // Количество блоков присвоенных фракции

// Перекраска ближайших блоков к главному зданию в цвет фракции

foreach (GameObject box in boxes)

{

if ( (Vector3.Distance(box.transform.position, boxes[randInt].transform.position) < 0.8f) & (box != boxes[randInt]) )

{

box.GetComponent<Renderer>().material.color = newColor;

blocks += 1;

Debug.Log(box.transform.position);

}

}

// Перекраска ближайших блоков в цвет фракции прилежищих к блокам уже перекрашенных в цвет фракции

if (blocks<9)

{

foreach (GameObject box1 in boxes)

{

foreach (GameObject box2 in boxes)

{

if ( (Vector3.Distance(box1.transform.position, box2.transform.position) < 0.8f) & (box2.GetComponent<Renderer>().material.color == newColor) & (box1.GetComponent<Renderer>().material.color != newColor))

{

box1.GetComponent<Renderer>().material.color = newColor;

blocks += 1;

Debug.Log(box1.transform.position);

break;

}

}

if (blocks >= 9)

{

break;

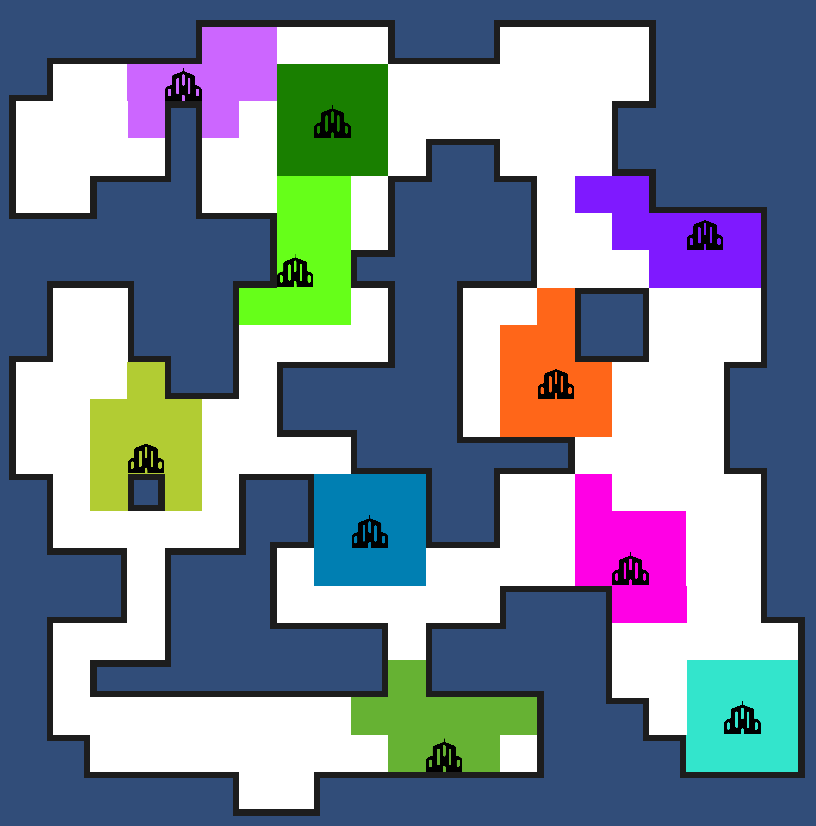
}

}

}

}

Результат при заданных 10 фракциях получается следующий:



### Движение камеры

Скрипт для движения камерой я нашёл в интернете:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CamEvents : MonoBehaviour

{

Vector3 touch;

public float sizeWidthTemp = 0;

public float sizeHeightTemp = 0;

public float zoomMin = 1;

public float zoomMax = 8;

public bool pause = false; // Переменная, с которой взаимодействует меню паузы

// Update is called once per frame

void Update()

{

if (!pause)

{

if (Input.GetMouseButtonDown(0))

{

touch = Camera.main.ScreenToWorldPoint(Input.mousePosition);

}

if (Input.touchCount == 2) // Приближение камеры

{

Touch touchZero = Input.GetTouch(0);

Touch touchOne = Input.GetTouch(1);

Vector2 touchZeroLastPos = touchZero.position - touchZero.deltaPosition;

Vector2 touchOneLastPos = touchOne.position - touchOne.deltaPosition;

float distTouch = (touchZeroLastPos - touchOneLastPos).magnitude;

float currentDistTouch = (touchZero.position - touchOne.position).magnitude;

float difference = currentDistTouch - distTouch;

zoom(difference \* 0.01f);

}

else if (Input.GetMouseButton(0)) // Движение камерой

{

Vector3 direction = touch - Camera.main.ScreenToWorldPoint(Input.mousePosition);

if (((Camera.main.transform.position.y + direction.y <= sizeHeightTemp) & (Camera.main.transform.position.x + direction.x <= sizeWidthTemp)) & ((Camera.main.transform.position.y + direction.y >= 0) & (Camera.main.transform.position.x + direction.x >= 0)))

{

Camera.main.transform.position += direction;

}

}

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

}

}

void zoom(float increment)

{

Camera.main.orthographicSize = Mathf.Clamp(Camera.main.orthographicSize - increment, zoomMin, zoomMax);

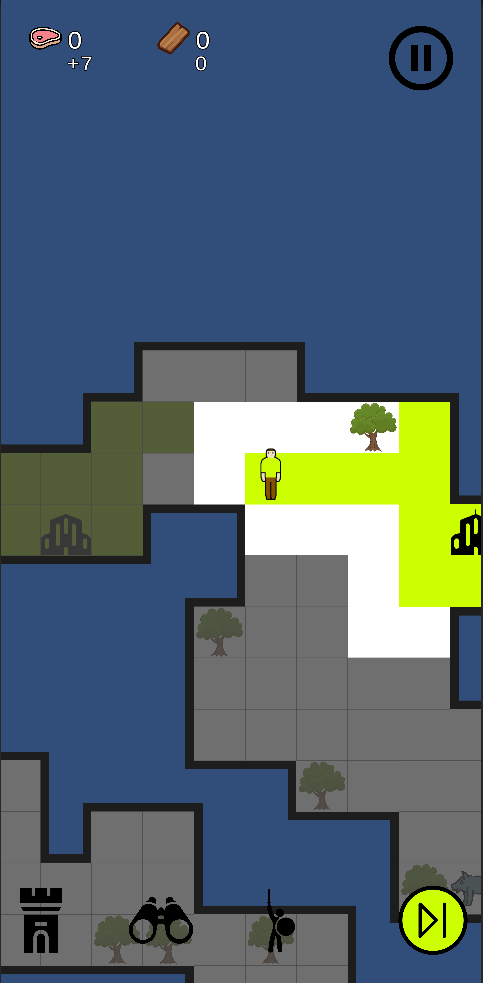
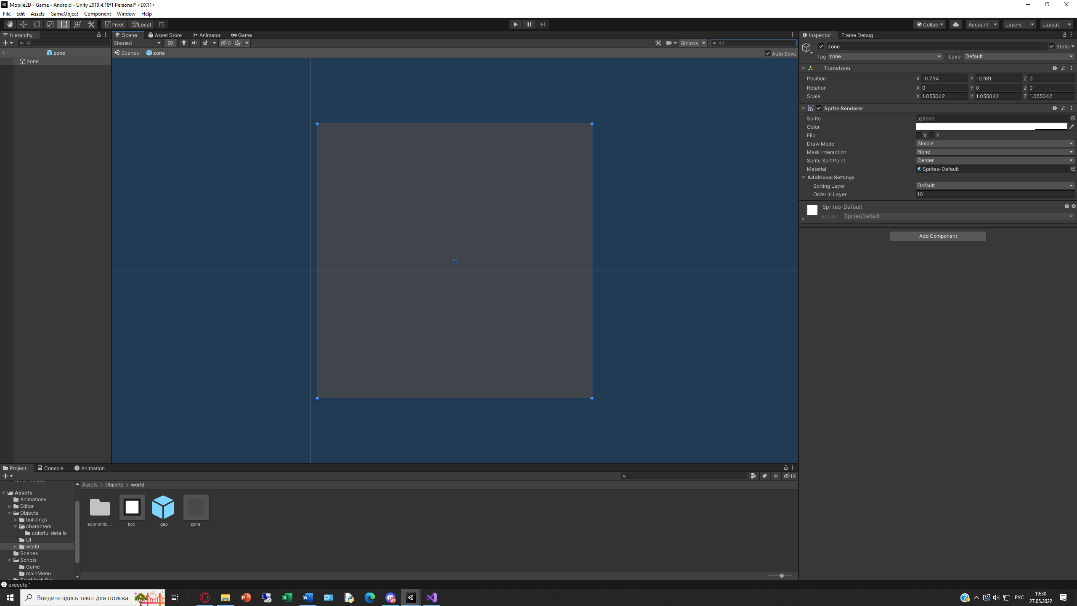
}

}

При одном касании вы можете передвигать камеру пальцем, двигая его по экрану, а приближать двумя касаниями

### Зоны

Где-то между фракциями и магазином возникают они – зоны.

Зоны – это декоративные префабы, которые нужны для визуального отображения блоков на которые невозможны: перемещение, спавн выбранного для покупки персонажа. Следовательно они используются при покупке и перемещении объектов.

Приведу скрипты, описывающие их использование:

На персонаже есть метод:

public void SeeZone()

{

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

for (int i = 0; i < boxes.Length; i++)

{

if (block == boxes[i])

{

availableBlocks.Add(boxes[i]);

}

else if (!CanTakeEnemyTerritory)

{

if (Vector3.Distance(boxes[i].transform.position, gameObject.transform.position) > movementDistanceInColor)

{

zoneBlocks.Add(boxes[i]);

}

else if (colorful\_details.GetComponent<Renderer>().material.color != boxes[i].GetComponent<Renderer>().material.color)

{

if (boxes[i].GetComponent<Renderer>().material.color == new Color(1, 1, 1, 1))

{

if (boxes[i].GetComponent<box>().character != null)

{

if ((power <= boxes[i].GetComponent<box>().character.GetComponent<character>().power) & (!highestPower) & !(CanKillSamePower & (power == boxes[i].GetComponent<box>().character.GetComponent<character>().power)))

{

zoneBlocks.Add(boxes[i]);

}

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((colorful\_details.GetComponent<Renderer>().material.color == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

}

/\*if (boxes[i].box.building != null)

{

}\*/

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((colorful\_details.GetComponent<Renderer>().material.color == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

} else { zoneBlocks.Add(boxes[i]); }

} else { availableBlocks.Add(boxes[i]); }

}

// Power > -1

else if ((colorful\_details.GetComponent<Renderer>().material.color == boxes[i].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[i].transform.position, gameObject.transform.position) > movementDistanceInColor))

{

zoneBlocks.Add(boxes[i]);

}

else if ((boxes[i].GetComponent<box>().character!=null) & (colorful\_details.GetComponent<Renderer>().material.color == boxes[i].GetComponent<Renderer>().material.color))

{

if ((boxes[i].GetComponent<box>().character.GetComponent<character>().id == id) & (!highestPower))

{

availableBlocks.Add(boxes[i]);

}

else

{

zoneBlocks.Add(boxes[i]);

}

}

else if (colorful\_details.GetComponent<Renderer>().material.color != boxes[i].GetComponent<Renderer>().material.color)

{

if (boxes[i].GetComponent<box>().character != null)

{

if ( (power <= boxes[i].GetComponent<box>().character.GetComponent<character>().power) & (!highestPower) & !(CanKillSamePower & (power == boxes[i].GetComponent<box>().character.GetComponent<character>().power)))

{

zoneBlocks.Add(boxes[i]);

}

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((colorful\_details.GetComponent<Renderer>().material.color == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

}

else if ( (!highestPower) & (power <= boxes[i].GetComponent<box>().power) & !(CanKillSamePower & (power == boxes[i].GetComponent<box>().power)))

{

zoneBlocks.Add(boxes[i]);

}

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((colorful\_details.GetComponent<Renderer>().material.color == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

} else { availableBlocks.Add(boxes[i]); }

}

for (int i = 0; i < zoneBlocks.Count; i++)

{

Instantiate(zone, zoneBlocks[i].transform.position, zoneBlocks[i].transform.rotation);

}

zoneBlocks.Clear();

}

Этот и подобные методы, создающие зону, проверяют множество условий и создают или не создают зону согласно им. Также они создают список блоков, доступных для использования, это создано с целью оптимизации.

Методы RemoveZone() как правило малонагруженны:

public void RemoveZone()

{

Debug.Log("nee");

GameObject[] zoneBlocksToDestroy = GameObject.FindGameObjectsWithTag("zone");

for (int i = 0; i < zoneBlocksToDestroy.Length; i++)

{

Destroy(zoneBlocksToDestroy[i]);

}

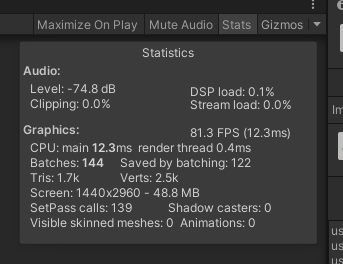
availableBlocks.Clear();

}

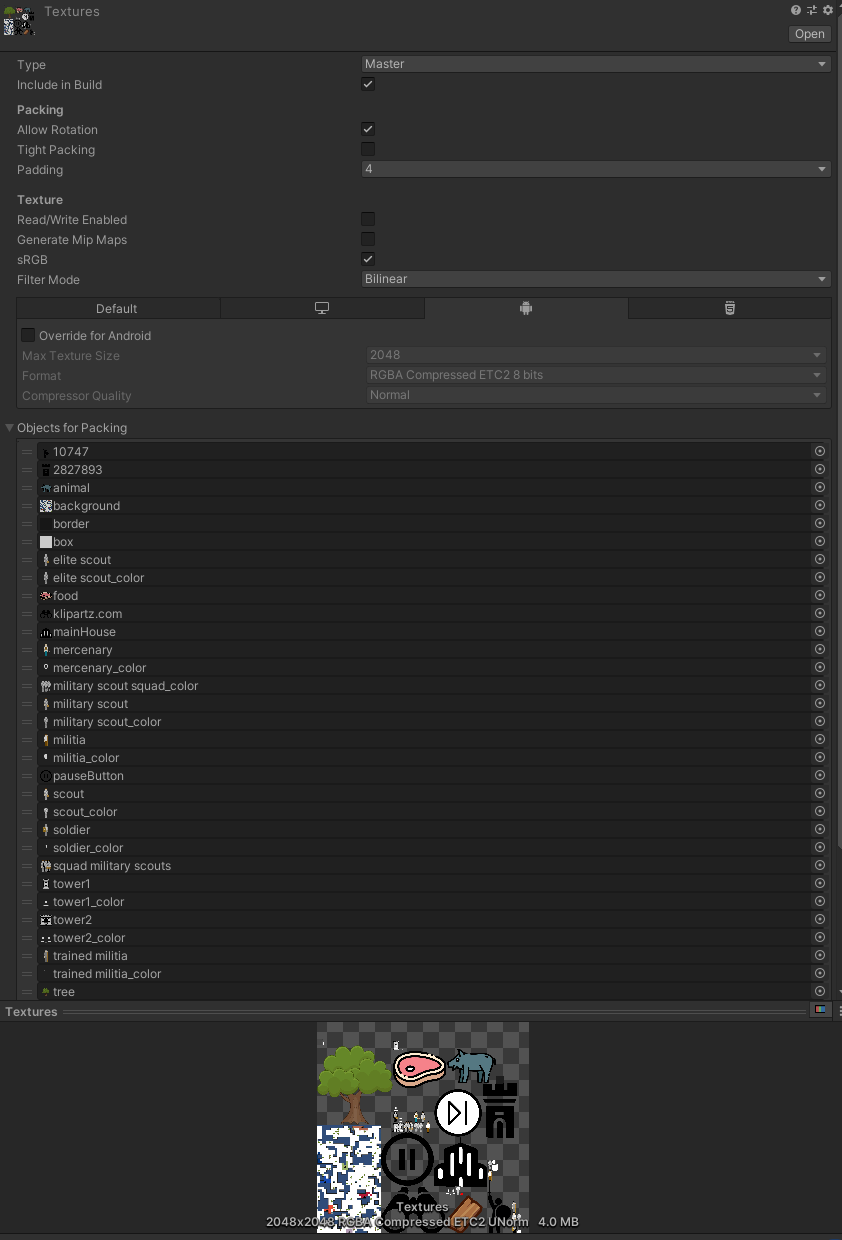
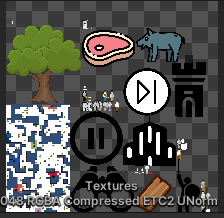
### Оптимизация

Мой проект был очень медленным на мобильном устройстве и я решил узнать об оптимизации в юнити. Для начала я отправился в настройки проекта и попытался снизить графику, но это не помогло, основная проблема была не в этом.

Для понимания проблемы стоит обратить на кнопки в правом верхнем углы, а именно на “stats”:

Здесь можно посмотреть статистику нагрузки проекта. Важно обратить внимание на пункт “Batches”. Он показывает количество запросов, отправленных графическому API (например, OpenGL или Direct3D). . Графический API производит значительную работу, что сильно влияет на производительность CPU. Насколько я помню в идеале лучше не превышать эту отметку в 300 запросов.

Самым простым решением проблему, будет создать атлас текстур:

  *\*текстурный атлас*

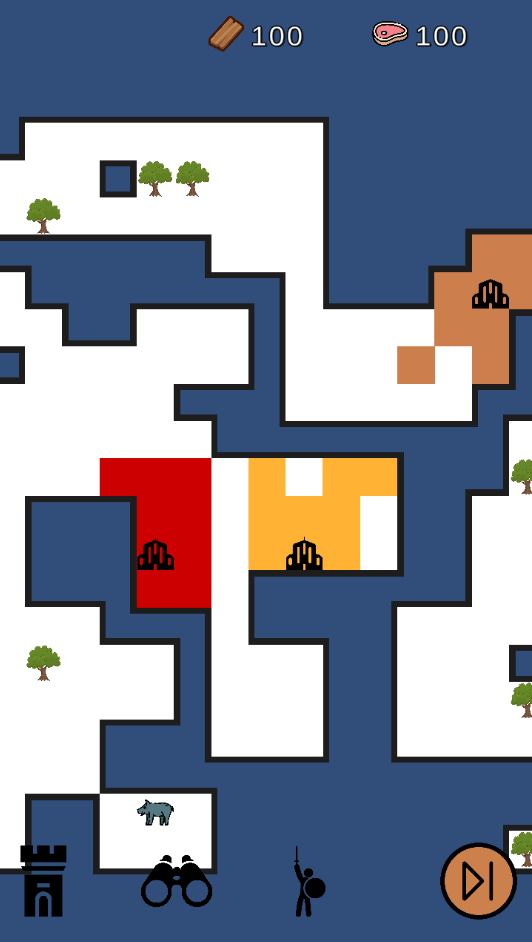
Атлас текстур позволяет загружать текстуры “пачками”, что значительно уменьшает количество батчей.

Ещё одной проблемой стало большое количество объектов на сцене. Юнити приходится для кажного из них проводить некие расчёты. У этой проблемы тоже есть решение – статичные объекты. Статичные объекты – статичны и им требуется значительно меньше расчётов. Если рассматривать игроговый результат без ограничения числа кадров, то прирост фпс на компьютере составил до 600 от предыдущих значений, на телефоне это невероятно ощутимо.

Ещё на производительность влияет количество сообщений выводящихся в консоли, но это я замечал только в движке, при бесконечном цикле. Как дело обстоит на телефоне без таких циклов, мне не известно. Поэтому я не спешу убирать все логи игры.

### Магазин

Думая что будет на данном этапе лучше: экономика или магазин, я решил сначала основать магазин. Для магазина было необходимо немного развить экономику, то есть добавить валюту и способы ещё отнятия. Валютой будут еда и материалы (дерево), а за манипуляции с валютами будет ответственен ранее упоминавшийся “my\_events” (не путать с EventSystem!). Он как и генератор уровня находится за зоной видимости камеры и также выглядит как красный квадрат.

Как видно на скриншоте, магазин представлен тремя кнопками внизу экрана: башня, бинокль и воин. Их названия: “BuildingButton”, “ColonistButton” и “UnitButton”. В кажном из них есть свой набор игровых объектов со своими назначениями. Здания – защищают территорию равную квадрату 3 на 3, где в центре находится здание, здания статичны (не могут перемещаться). Разведчики (колонисты, скауты) – воины живущие лишь до тех пор, пока они не сделают ход, обладают большим радиусом для хода вне территории фракции. Воины (армии) – стандартный представитель управляемых персонажей. Каждый игровой персонаж имеет свою силу, от которой зависит их цена. Общее название для разведчиков и воинов – юниты.

Давайте начнём. Кнопки для магазина я показал, ещё нужен банк валют. То есть визуальное отображение количество ресурсов, которыми владеет игрок (фракция)

*\*Банк валют*

Чтобы управлять накоплениями нам хватит следующих функций в “my\_events”:

using System.Collections;

using System.Collections.Generic;

using System;

using UnityEngine;

using TMPro;

using UnityEngine.UI;

public class my\_events : MonoBehaviour

{

public void plusFood(int count)

{

int newValue = queueFood[ind] + count;

Debug.Log(newValue);

queueFood[ind] = newValue;

food.text = newValue.ToString();

}

public bool checkFoodMinus(int count) // Проверка, можно ли отнять данное количество еды

{

int newValue = queueFood[ind] - count;

if (newValue >= 0)

{

return true;

}

else

{

return false;

}

}

public void minusFood(int count)

{

int newValue = queueFood[ind] - count;

if (newValue >= 0)

{

queueFood[ind] = newValue;

food.text = newValue.ToString();

}

}

public void plusWood(int count)

{

int newValue = queueWood[ind] + count;

Debug.Log(newValue);

queueWood[ind] = newValue;

wood.text = newValue.ToString();

}

public bool checkWoodMinus(int count) // Проверка, можно ли отнять данное количество ресурсов

{

int newValue = queueWood[ind] - count;

if (newValue >= 0)

{

return true;

}

else

{

return false;

}

}

public void minusWood(int count)

{

int newValue = queueWood[ind] - count;

if (newValue >= 0)

{

queueWood[ind] = newValue;

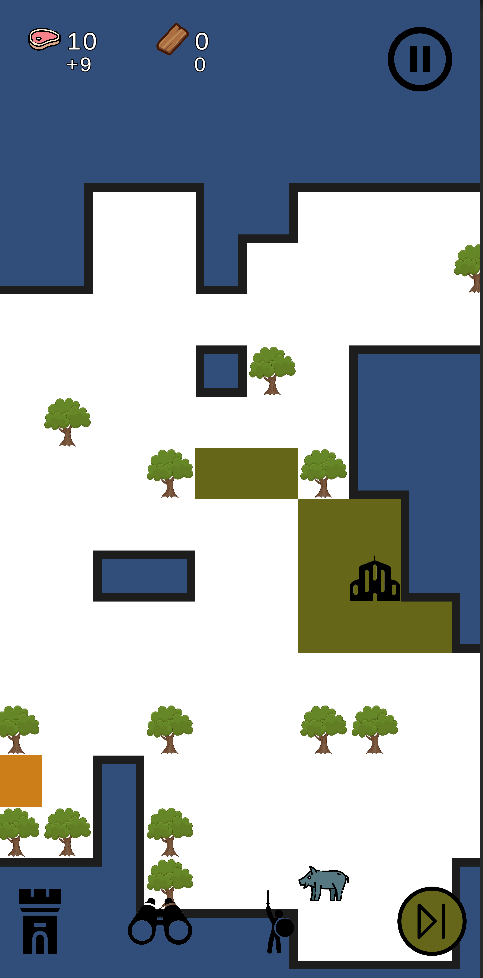
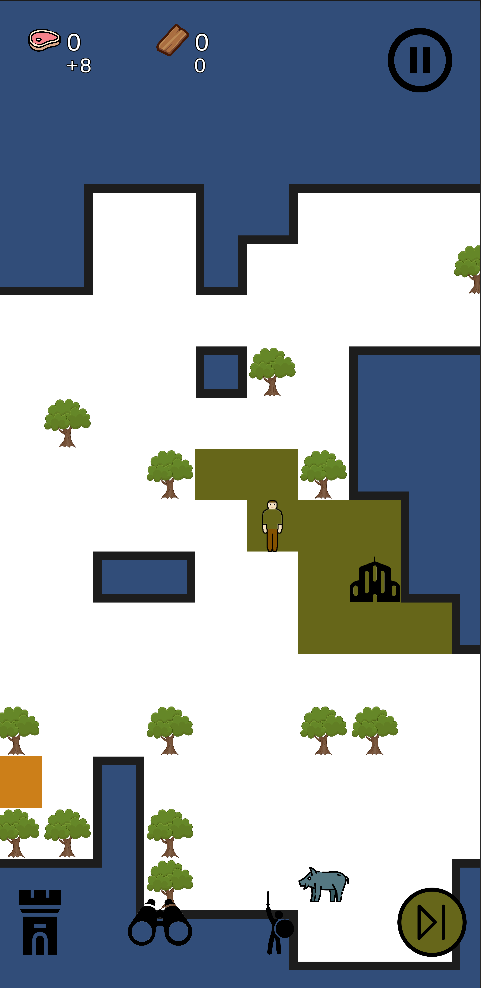
wood.text = newValue.ToString();

}

}

*\*wood и food – переменные, представляющие собой текст из банка валют*

Далее нужны скрипты на магазины. Магазин будет показывать, где можно разместить юнита и показывать его на экране. Цель магазина - указать блокам, какого персонажа следует инициировать и где именно. Значит нужно добавить новые функции для блоков. Указания будут выражены через тэги объектов, которые будут визуально показаны на сцене, а также данных указанных в объекте. Продемонстрирую работу данного кода и сам код:

1)  2)  3) 

*\*1 Начальное состояние сцены*

*\*2 Выбор юнита в магазине, нажатием несколько раз на кнопку магазина*

*\*3 Спавн юнита*

Реализация в коде:

Магазин (практически идентичен на всех трёх кнопках):

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UnitButtonScript : MonoBehaviour

{

public GameObject zone;

public List<GameObject> availableBlocks = new List<GameObject>();

List<GameObject> zoneBlocks = new List<GameObject>();

// Визуально показать, где можно разместить персонажа, с помощью зоны

public void SeeZone()

{

my\_events my\_events = GameObject.FindGameObjectsWithTag("my\_events")[0].GetComponent<my\_events>();

//my\_events.turn

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

character subject = GameObject.FindGameObjectsWithTag("buy\_me")[0].GetComponent<whoAmI>().Me.GetComponent<character>();

for (int i = 0; i < boxes.Length; i++)

{

if ((boxes[i].GetComponent<box>().character != null) & (my\_events.turn == boxes[i].GetComponent<Renderer>().material.color))

{

if ((boxes[i].GetComponent<box>().character.GetComponent<character>().id == subject.id) & (!subject.highestPower))

{

availableBlocks.Add(boxes[i]);

}

else

{

zoneBlocks.Add(boxes[i]);

}

}

else if (my\_events.turn != boxes[i].GetComponent<Renderer>().material.color)

{

if (boxes[i].GetComponent<box>().character != null)

{

if ((subject.power <= boxes[i].GetComponent<box>().character.GetComponent<character>().power) & (!subject.highestPower) & !(subject.CanKillSamePower & (subject.power == boxes[i].GetComponent<box>().character.GetComponent<character>().power)))

{

zoneBlocks.Add(boxes[i]);

}

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((my\_events.turn == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < subject.movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

}

else if ((!subject.highestPower) & (subject.power <= boxes[i].GetComponent<box>().power) & !(subject.CanKillSamePower & (subject.power == boxes[i].GetComponent<box>().power)))

{

zoneBlocks.Add(boxes[i]);

}

else

{

bool placeZone = true;

for (int j = 0; j < boxes.Length; j++)

{

if ((my\_events.turn == boxes[j].GetComponent<Renderer>().material.color) & (Vector3.Distance(boxes[j].transform.position, boxes[i].transform.position) < subject.movementDistanceOutColor))

{

placeZone = false;

break;

}

}

if (placeZone) { zoneBlocks.Add(boxes[i]); } else { availableBlocks.Add(boxes[i]); }

}

}

else { availableBlocks.Add(boxes[i]); }

}

for (int i = 0; i < zoneBlocks.Count; i++)

{

Instantiate(zone, zoneBlocks[i].transform.position, zoneBlocks[i].transform.rotation);

}

zoneBlocks.Clear();

}

// Убрать зону

public void RemoveZone()

{

Debug.Log("nee");

GameObject[] zoneBlocksToDestroy = GameObject.FindGameObjectsWithTag("zone");

for (int i = 0; i < zoneBlocksToDestroy.Length; i++)

{

Destroy(zoneBlocksToDestroy[i]);

}

availableBlocks.Clear();

}

// Вернуть магазин в изначальное состояние

public void Bought()

{

RemoveZone();

Debug.Log("execute " + pressNow.ToString());

GameObject[] units = GameObject.FindGameObjectsWithTag("buy\_me");

if (units.Length > 0)

{

moveDown(units[0]);

units[0].tag = "unitToBuy";

pressNow = 0;

}

}

// Выбор персонажа для покупки

int pressNow = 0;

public void ButtonPress()

{

RemoveZone();

GameObject ColonistButton = GameObject.FindGameObjectsWithTag("ColonistButton")[0];

ColonistButton.GetComponent<ColonistButtonScript>().Bought();

GameObject BuildingButton = GameObject.FindGameObjectsWithTag("BuildingButton")[0];

BuildingButton.GetComponent<BuildingButtonScript>().Bought();

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected");

if (selected.Length>0)

{

selected[0].tag = "character";

}

GameObject[] unit\_buy\_me = GameObject.FindGameObjectsWithTag("buy\_me");

if (unit\_buy\_me.Length > 0)

{

unit\_buy\_me[0].tag = "unitToBuy";

moveDown(unit\_buy\_me[0]);

}

pressNow += 1;

GameObject[] units = GameObject.FindGameObjectsWithTag("unitToBuy");

Debug.Log(pressNow);

bool reset = true;

GameObject buy\_me = gameObject;

foreach (GameObject unit in units)

{

if (unit.GetComponent<whoAmI>().place == pressNow )

{

buy\_me = unit;

reset = false;

break;

}

}

if (reset)

{

Debug.Log("Reset");

pressNow = 0;

} else

{

buy\_me.tag = "buy\_me";

moveUp(buy\_me);

}

}

// Показать выбранного персонажа для покупки

void moveUp(GameObject unit)

{

my\_events my\_events = GameObject.FindGameObjectsWithTag("my\_events")[0].GetComponent<my\_events>();

unit.GetComponent<whoAmI>().MyColorDetails.GetComponent<Image>().color = my\_events.turn;

moveDown(unit);

unit.GetComponent<whoAmI>().playAnimIn(); // Анимация выхода персонажа на сцену

unit.tag = "buy\_me";

SeeZone();

}

// Скрыть персонажа

void moveDown(GameObject unit)

{

unit.GetComponent<whoAmI>().undoAnim(); // Перемещение персонажа за сцену

}

}

Магазин использует специальные тэги (например “buy\_me”) для указания игрового персонажа, которого вы выбрали для покупки и создаёт список доступных блоков, на которые можно установить персонажа.

Вы заметили обращение к скрипту “whoAmI”? Он наложен на персонажах, которые визуально показывают кого вы выбрали для покупки. Одного из них вы могли видеть на скриншоте, где демонстрировалась работа магазина. Именно там прописаны необходимые для спавна юнита данные:

public class whoAmI : MonoBehaviour

{

public GameObject Me; // Юнит для спавна

public GameObject MyColorDetails; // Цветные детали объекта, не относится к “Me”

public int priceFood;

public int priceWood;

public int place;

bool play = false; // Проиграть ли анимацию

float startY; // Начальная позиция по Y

// Переменные для работы с анимацией

float frames = 10;

float frameNow = 0;

float deltaPos = 1000;

public void Start() // Расчёт начальной позиции по Y

{

startY = (Screen.height \* 2.5f) / 15 - 1000;

if (Screen.height >= 1280)

{

startY = (Screen.height \* 2.5f) / 15 - (1000 \* (Screen.height / 1280));

}

else

{

startY = (Screen.height \* 2.5f) / 15 - (1000 \* (Screen.height \* 0.00078125f));

}

}

public void playAnimIn() // Проиграть анимацию

{

play = true;

}

public void undoAnim() // Увести персонажа за сцену

{

play = false;

if (Screen.height >= 1280)

{

transform.position = new Vector3((Screen.width \* 6) / 12, startY - (1000 \* (Screen.height / 1280)), 0);

}

else

{

transform.position = new Vector3((Screen.width \* 6) / 12, startY - (1000 \* (Screen.height \* 0.00078125f)), 0);

}

}

void Update() // Анимация

{

if (play)

{

if (frameNow > frames)

{

frameNow = 0;

play = false;

}

else

{

if (Screen.height >= 1280)

{

transform.position = new Vector3((Screen.width \* 6) / 12, startY + ((frameNow / frames) \* deltaPos \* (Screen.height / 1280)), 0);

}

else

{

transform.position = new Vector3((Screen.width \* 6) / 12, startY + ((frameNow / frames) \* deltaPos \* (Screen.height \* 0.00078125f)), 0);

}

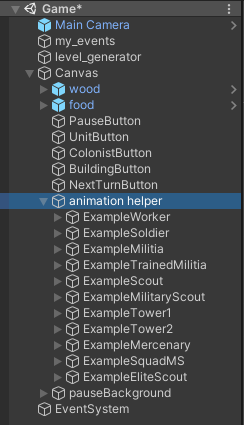
frameNow += 1;

}

}

}

}

На будущее стоит отметить, что каждый такой объект имеет привязку положения к “animation helper”. Это пустой объект, который является вспомогательным для анимации.

Дальше работу берёт на себя доступный блок:

// Заключение кнопок магазина в переменные

GameObject UnitButton = GameObject.FindGameObjectsWithTag("UnitButton")[0];

GameObject ColonistButton = GameObject.FindGameObjectsWithTag("ColonistButton")[0];

GameObject buildingButton = GameObject.FindGameObjectsWithTag("BuildingButton")[0];

// Входит ли блок хотя бы в один из списков доступных, для размещения купленного персонажа

if (UnitButton.GetComponent<UnitButtonScript>().availableBlocks.Contains(gameObject) | ColonistButton.GetComponent<ColonistButtonScript>().availableBlocks.Contains(gameObject) | buildingButton.GetComponent<BuildingButtonScript>().availableBlocks.Contains(gameObject))

{

// Получение массивов с выбранными для покупки юнитами и разведчиками

GameObject[] unit\_buy\_me = GameObject.FindGameObjectsWithTag("buy\_me");

GameObject[] colonist\_buy\_me = GameObject.FindGameObjectsWithTag("colonist\_buy\_me");

// Работа с выбранным юнитом, если он есть

if (unit\_buy\_me.Length > 0)

{

Debug.Log("unit\_buy\_me.Length > 0");

if (UnitButton.GetComponent<UnitButtonScript>().availableBlocks.Contains(gameObject))

{

whoAmI characteristics = unit\_buy\_me[0].GetComponent<whoAmI>(); // Данные о выбранном юните

bool canBuy = (my\_events.checkFoodMinus(characteristics.priceFood) & my\_events.checkWoodMinus(characteristics.priceWood)); // Можно ли купить юнита

if (canBuy)

{

// Забор валют

my\_events.minusFood(characteristics.priceFood);

my\_events.minusWood(characteristics.priceWood);

GameObject unitToSpawn = characteristics.Me;

Debug.Log(character);

Debug.Log(transform.position);

Debug.Log(character != null);

Debug.Log(gameObject.GetComponent<Renderer>().material.color == my\_events.turn);

// Если на блоке есть персонаж, то удалить его и инициировать улучшенную версию выбранного игрового объекта

if ((character != null) & (gameObject.GetComponent<Renderer>().material.color == my\_events.turn))

{

Debug.Log("I AM HEREEEEEEEEEEEEEEE");

bool CanWalk = character.GetComponent<character>().CanWalk;

unitToSpawn = Instantiate(character.GetComponent<character>().upgrade, transform.position, transform.rotation);

unitToSpawn.GetComponent<character>().SetColor(my\_events.turn);

unitToSpawn.GetComponent<character>().block = gameObject;

character.tag = "Destroyed";

Destroy(character);

if (!CanWalk)

{

unitToSpawn.GetComponent<character>().SetNotWalk();

unitToSpawn.transform.position = transform.position;

}

character = unitToSpawn;

unitToSpawn.tag = "character";

ChangeColor(gameObject.GetComponent<Renderer>().material.color, my\_events.turn);

}

// Иначе просто создать выбранного юнита на блоке

else

{

unitToSpawn = Instantiate(unitToSpawn, transform.position, transform.rotation);

unitToSpawn.GetComponent<character>().SetColor(my\_events.turn);

unitToSpawn.GetComponent<character>().block = gameObject;

if (character != null)

{

Destroy(character);

}

if ((building != null) & (gameObject.GetComponent<Renderer>().material.color != my\_events.turn))

{

if (building.tag != "MainHouse")

{

building.GetComponent<BuildingScript>().Weakness();

}

}

character = unitToSpawn;

unitToSpawn.tag = "character";

if (gameObject.GetComponent<Renderer>().material.color != my\_events.turn)

{

character.GetComponent<character>().SetNotWalk();

}

ChangeColor(gameObject.GetComponent<Renderer>().material.color, my\_events.turn);

}

UnitButton.GetComponent<UnitButtonScript>().Bought();

}

else { UnitButton.GetComponent<UnitButtonScript>().Bought(); }

}

else { UnitButton.GetComponent<UnitButtonScript>().Bought(); }

}

else if (character == null) // Если на блоке нет персонажей

{

// Схожий алгоритм с покупкой юнитов, но без улучшений

if (colonist\_buy\_me.Length > 0)

{

if (my\_events.turn == gameObject.GetComponent<Renderer>().material.color)

{

whoAmI characteristics = colonist\_buy\_me[0].GetComponent<whoAmI>();

bool canBuy = (my\_events.checkFoodMinus(characteristics.priceFood) & my\_events.checkWoodMinus(characteristics.priceWood));

if (gameObject.GetComponent<Renderer>().material.color == my\_events.turn)

{

if (canBuy)

{

my\_events.minusFood(characteristics.priceFood);

my\_events.minusWood(characteristics.priceWood);

GameObject colonistToSpawn = characteristics.Me;

colonistToSpawn = Instantiate(colonistToSpawn, transform.position, transform.rotation);

colonistToSpawn.GetComponent<character>().SetColor(my\_events.turn);

colonistToSpawn.GetComponent<character>().block = gameObject;

character = colonistToSpawn;

colonistToSpawn.tag = "character";

ColonistButton.GetComponent<ColonistButtonScript>().Bought();

}

else { ColonistButton.GetComponent<ColonistButtonScript>().Bought(); }

}

else { ColonistButton.GetComponent<ColonistButtonScript>().Bought(); }

}

else { ColonistButton.GetComponent<ColonistButtonScript>().Bought(); }

}

// Схожий алгоритм с покупкой юнитов, но без улучшений

else if (building == null)

{

GameObject[] building\_buy\_me = GameObject.FindGameObjectsWithTag("building\_buy\_me");

if (building\_buy\_me.Length > 0)

{

if (my\_events.turn == gameObject.GetComponent<Renderer>().material.color)

{

whoAmI characteristics = building\_buy\_me[0].GetComponent<whoAmI>();

bool canBuy = (my\_events.checkFoodMinus(characteristics.priceFood) & my\_events.checkWoodMinus(characteristics.priceWood));

if (gameObject.GetComponent<Renderer>().material.color == my\_events.turn)

{

if (canBuy)

{

my\_events.minusFood(characteristics.priceFood);

my\_events.minusWood(characteristics.priceWood);

GameObject buildingToSpawn = characteristics.Me;

buildingToSpawn = Instantiate(buildingToSpawn, transform.position, transform.rotation);

buildingToSpawn.GetComponent<BuildingScript>().SetColor(my\_events.turn);

building = buildingToSpawn;

buildingToSpawn.GetComponent<BuildingScript>().block = gameObject;

buildingToSpawn.GetComponent<BuildingScript>().Defend();

buildingButton.GetComponent<BuildingButtonScript>().Bought();

}

else { buildingButton.GetComponent<BuildingButtonScript>().Bought(); }

}

else { buildingButton.GetComponent<BuildingButtonScript>().Bought(); }

}

else { buildingButton.GetComponent<BuildingButtonScript>().Bought(); }

}

}

}

}

### Юниты и здания

Во время разработки магазина, были разработаны и персонажи, которыми пользуется игрок:

У разведчиков и воинов один код, но он использует разные значения переменных из-за чего создаётся разное поведение, в одном скрипте. У башен (зданий) же скрипт отдельный. В нём описывается защита территорий, которая реализована в блоках, в виде силы блоков и обновление защиты, для адаптации под изменения поля боя. Давайте покажу сам код:

На воинах/разведчиках находится скрипт:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class character : MonoBehaviour

{

public int id; // id персонажа

public GameObject upgrade; // кем станет персонаж при улучшении

public GameObject block;

public GameObject zone;

public GameObject colorful\_details; // цветная часть персонажа

public float movementDistanceInColor = 2.5f; // Дистанция перемещения внутри блоков фракции

public float movementDistanceOutColor = 0.8f; // Дистанция перемещения вне блоков фракции

public int power = 1;

public bool DestroyAfterStep = false;

public bool CanTakeEnemyTerritory = true;

public bool CanKillSamePower = false;

public bool highestPower = false; // Юнит является обладателем высшей силы из всех

public List<GameObject> availableBlocks = new List<GameObject>(); // Куда юнит может походить

public bool CanWalk = true; // Может ли ходить

public Vector3 startPos; // Начальная позиция анимации

int frames = 56; // Количество кадров в анимации

float deltapos = 0.08f; // Смещение от начальной позиции при анимации

int nowFrame = 0; // Проигрываемый кадр анимации

List<GameObject> zoneBlocks = new List<GameObject>(); // Куда будут установленны блоки зоны

void Start()

{

startPos = transform.position;

}

public void SetColor(Color colorToSet) // Установка цветных деталей

{

colorful\_details = Instantiate(colorful\_details, transform.position, transform.rotation);

colorful\_details.transform.parent = gameObject.transform;

colorful\_details.GetComponent<colorForUnit>().Parent = gameObject;

colorful\_details.GetComponent<Renderer>().material.color = colorToSet;

}

public void SetNotWalk() // Персонаж походил

{

transform.position = startPos;

CanWalk = false;

transform.position = block.transform.position;

}

public void SetWalk() // Персонаж снова может ходить

{

startPos = transform.position;

CanWalk = true;

}

… SeeZone() … RemoveZone() …

public void OnMouseDown()

{

if (block==null)

{

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

foreach (GameObject box in boxes)

{

if (box.transform.position == transform.position)

{

block = box;

box.GetComponent<box>().character = gameObject;

break;

}

}

}

else if (block.GetComponent<box>().character==null)

{

block.GetComponent<box>().character = gameObject;

}

my\_events my\_events = GameObject.FindGameObjectsWithTag("my\_events")[0].GetComponent<my\_events>();

GameObject[] unit\_buy\_me = GameObject.FindGameObjectsWithTag("buy\_me");

/\*if (unit\_buy\_me.Length > 0)

{

unit\_buy\_me[0].transform.position = new Vector3(unit\_buy\_me[0].transform.position.x, -550, unit\_buy\_me[0].transform.position.z);

unit\_buy\_me[0].tag = "unitToBuy";

RemoveZone();

}\*/

block.GetComponent<box>().OnMouseDown();

}

void Update() // Анимация доступности ходьбы

{

if (CanWalk)

{

if (nowFrame == frames)

{

nowFrame = 0;

}

else if (nowFrame <= frames / 2)

{

float y = startPos.y + deltapos \* (nowFrame / (frames \* 0.5f));

transform.position = new Vector3(startPos.x, y, startPos.z);

} else

{

float y = startPos.y + deltapos - deltapos \* ((nowFrame - frames \* 0.5f) / (frames \* 0.5f));

transform.position = new Vector3(startPos.x, y, startPos.z);

}

nowFrame += 1;

}

}

}

На зданиях наложен следующий скрипт:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BuildingScript : MonoBehaviour

{

// Start is called before the first frame update

public List<GameObject> DefendBlocks = new List<GameObject>();

public GameObject block;

public GameObject colorful\_details;

public int power;

public void SetColor(Color colorToSet) // Установка цветных деталей

{

colorful\_details = Instantiate(colorful\_details, transform.position, transform.rotation);

colorful\_details.transform.parent = gameObject.transform;

colorful\_details.GetComponent<Renderer>().material.color = colorToSet;

}

public void Defend() // Защита доступных блоков

{

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

foreach (GameObject box in boxes)

{

if (Vector3.Distance(box.transform.position, gameObject.transform.position) < 0.8f)

{

if ( (box.GetComponent<box>().power<=power) & (box.GetComponent<Renderer>().material.color == colorful\_details.GetComponent<Renderer>().material.color) )

{

box.GetComponent<box>().power = power;

}

DefendBlocks.Add(box);

}

}

}

public void DefendAgain()

{

foreach (GameObject box in DefendBlocks)

{

if ((box.GetComponent<box>().power <= power) & (box.GetComponent<Renderer>().material.color == colorful\_details.GetComponent<Renderer>().material.color))

{

box.GetComponent<box>().power = power;

}

}

}

public void Weakness() // Убрать защиту с защищаемых блоков

{

gameObject.tag = "Destroyed";

foreach (GameObject box in DefendBlocks)

{

box.GetComponent<box>().power = -2;

}

GameObject[] buildings = GameObject.FindGameObjectsWithTag("building");

foreach (GameObject building in buildings)

{

building.GetComponent<BuildingScript>().DefendAgain();

}

Destroy(gameObject);

}

void OnMouseDown()

{

block.GetComponent<box>().OnMouseDown();

}

}

Следует пояснить что такое “colorful\_details”. Это “цветные детали”, то есть текстуры, наложенные поверх объектов, которым они принадлежат и используют цвет фракции к которой относится игровой объект. Они есть как у персонажей, зданий, так и у представлений покупаемых персонажей в магазине.

### Ходы и экономика

#### Переход хода и очереди

Самое время создать событие следующего хода. Читая последние скрипты, вы заметили, что там используются незнакомые для “моих событий” методы и свойства. Сейчас вы узнаете, что они значат. Для него создадим кнопку в левом нижнем углу экрана и займёмся ранее упоминавшимся “my\_events”:

Данная кнопка будет активировать определённый метод с скрипте моих событий. Она принимает цвет той фракции, которой принадлежит ход.

*\*Кнопка следующего хода*

Для начала работы всей системы смены хода необходимо с чего-то начать. Задать очередь, начальные значения валют. А ещё интерфейс игры на данный момент плохо масштабируемый и изменяется только по размеру, по свойству юнити. Это необходимо изменить

Из постоянного я добавлю, для баланса сил, после каждого хода, за всех персонажей во владении фракции, будет отниматься определённое количество еды. При невозможности выплатить долг, армия будет уничтожаться.

И вот код старта игры:

public class my\_events : MonoBehaviour

{

public GameObject PauseBackground;

public TextMeshProUGUI food;

public TextMeshProUGUI wood;

public Color turn;

public List<Color> queue = new List<Color>();

List<int> queueFood = new List<int>();

List<int> queueWood = new List<int>();

int ind = 0;

public int startPackFood = 10;

public int startPackWood = 0;

public int rewardFoodForBlock = 1;

public int rewardWoodForBlock = 0;

public int rewardForForest = 1;

public int rewardForAnimals = 1;

public int sanctionForUnit = 1;

int rewardFood = 0;

int rewardWood = 0;

void Start()

{

Application.targetFrameRate = 60; // Ограничение кадров в 60 фпс

}

public void start\_game(GameObject[] factions, Color col)

{

Debug.Log("Screen size:");

Debug.Log("==============");

Debug.Log("width:");

Debug.Log(Screen.width);

Debug.Log("height:");

Debug.Log(Screen.height);

Debug.Log("==============");

// Настройка положения объектов интерфейса пользователя

if (Screen.height >= 1280)

{

GameObject.FindGameObjectsWithTag("animation helper")[0].transform.position = new Vector3((Screen.width \* 6) / 12, ((Screen.height \* 2.5f) / 15) - 1000 \* (Screen.height / 1280), 0);

}

else

{

GameObject.FindGameObjectsWithTag("animation helper")[0].transform.position = new Vector3((Screen.width \* 6) / 12, ((Screen.height \* 2.5f) / 15) - 1000 \* (Screen.height \* 0.00078125f), 0);

}

PauseBackground.transform.position = new Vector3(PauseBackground.transform.position.x, PauseBackground.transform.position.y + (1000 \* Screen.height), PauseBackground.transform.position.z);

GameObject PauseButton = GameObject.FindGameObjectsWithTag("PauseButton")[0];

PauseButton.transform.position = new Vector3((Screen.width \* 10.5f) / 12, (Screen.height \* 14.1f) / 15, PauseButton.transform.position.z);

GameObject UnitButton = GameObject.FindGameObjectsWithTag("UnitButton")[0];

UnitButton.transform.position = new Vector3((Screen.width \* 7) / 12, Screen.height / 15, UnitButton.transform.position.z);

GameObject ColonistButton = GameObject.FindGameObjectsWithTag("ColonistButton")[0];

ColonistButton.transform.position = new Vector3((Screen.width \* 4) / 12, Screen.height / 15, ColonistButton.transform.position.z);

GameObject BuildingButton = GameObject.FindGameObjectsWithTag("BuildingButton")[0];

BuildingButton.transform.position = new Vector3((Screen.width \* 1) / 12, Screen.height / 15, BuildingButton.transform.position.z);

GameObject NextTurnButton = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

NextTurnButton.transform.position = new Vector3((Screen.width \* 10.8f) / 12, Screen.height / 15, NextTurnButton.transform.position.z);

GameObject foodGameO = GameObject.FindGameObjectsWithTag("food")[0];

foodGameO.transform.position = new Vector3((Screen.width \* 2.9f) / 12, (Screen.height \* 14.4f) / 15, food.transform.position.z);

GameObject woodGameO = GameObject.FindGameObjectsWithTag("wood")[0];

woodGameO.transform.position = new Vector3((Screen.width \* 6.1f) / 12, (Screen.height \* 14.4f) / 15, food.transform.position.z);

turn = col;

foreach (GameObject faction in factions) // Создание очереди и запись банков валют фракций

{

queue.Add(faction.GetComponent<Renderer>().material.color);

queueFood.Add(startPackFood);

queueWood.Add(startPackWood);

}

// Присваивание банку валют значений первой в очереди фракции

food.text = startPackFood.ToString();

wood.text = startPackWood.ToString();

// Перекраска кнопки следующего хода

GameObject but = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

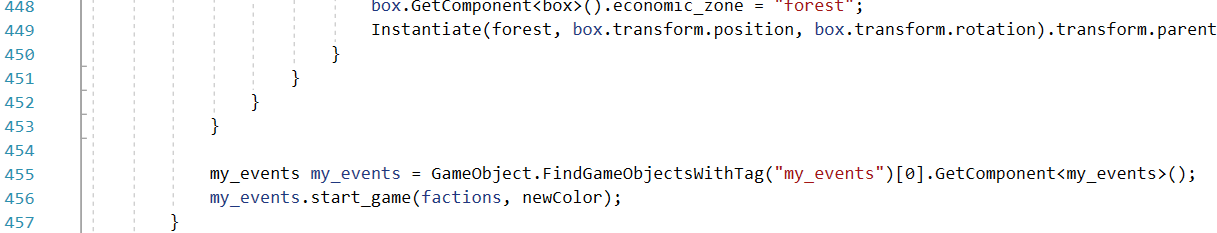
but.GetComponent<Image>().color = factions[0].GetComponent<Renderer>().material.color;

turn = factions[0].GetComponent<Renderer>().material.color;

}

}

Метод “start\_game” использует генератор уровня после окончания создания уровня:



А следующая часть отвечает за смену хода:

public void next\_turn()

{

// Если осталась одна фракция, выйти в главное меню

if (queue.Count<=1)

{

SceneManager.LoadScene("mainMenu");

}

else

{

// Установить все магазины в изначальное положение

GameObject.FindGameObjectsWithTag("UnitButton")[0].GetComponent<UnitButtonScript>().Bought();

GameObject.FindGameObjectsWithTag("ColonistButton")[0].GetComponent<ColonistButtonScript>().Bought();

GameObject.FindGameObjectsWithTag("BuildingButton")[0].GetComponent<BuildingButtonScript>().Bought();

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

GameObject[] characters = GameObject.FindGameObjectsWithTag("character");

GameObject[] buildings = GameObject.FindGameObjectsWithTag("building");

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected");

// Снять все выделения

if (selected.Length > 0)

{

selected[0].GetComponent<character>().RemoveZone();

selected[0].tag = "character";

}

// Если фракция не последняя в очереди

if (ind + 1 < queue.Count)

{

queueFood[ind] = Convert.ToInt32(food.text) + rewardFood;

queueWood[ind] = Convert.ToInt32(wood.text) + rewardWood;

// Манипуляции с персонажами

foreach (GameObject character in characters)

{

// Для фракции наступающего хода

if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == queue[ind + 1])

{

// Если не смогли оплатить долг, уничножить армию

if (queueFood[ind + 1] < 0)

{

character.GetComponent<character>().tag = "Destroy";

Destroy(character);

}

else { character.GetComponent<character>().SetWalk(); Debug.Log("Ты ходишь"); }

}

// Для ходившей фракции, запретить всем юнитам ходить

else if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == queue[ind])

{

character.GetComponent<character>().SetNotWalk();

Debug.Log("Ты не ходишь");

}

}

// Смена цвета, банка валют под другую фракцию

turn = queue[ind + 1];

food.text = queueFood[ind + 1].ToString();

wood.text = queueWood[ind + 1].ToString();

GameObject but = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

but.GetComponent<Image>().color = turn;

ind += 1;

}

else // Если фракция последняя в очереди

{

queueFood[ind] = Convert.ToInt32(food.text) + rewardFood;

queueWood[ind] = Convert.ToInt32(wood.text) + rewardWood;

foreach (GameObject character in characters)

{

if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == queue[0])

{

if (queueFood[0] < 0)

{

character.GetComponent<character>().tag = "Destroy";

Destroy(character);

}

else { character.GetComponent<character>().SetWalk(); Debug.Log("Ты ходишь"); }

}

else if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == queue[ind])

{

Debug.Log("Ты не ходишь");

Debug.Log(character.transform.position);

character.GetComponent<character>().SetNotWalk();

}

}

ind = 0;

turn = queue[0];

food.text = queueFood[ind].ToString();

wood.text = queueWood[ind].ToString();

GameObject but = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

but.GetComponent<Image>().color = turn;

}

foreach (GameObject building in buildings) // Перенастроить защиту зданий

{

building.GetComponent<BuildingScript>().DefendAgain();

}

foreach (GameObject character in characters) // Для следующей очереди фракции, разрешить ходить юнитам

{

if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == queue[ind])

{

character.GetComponent<character>().SetWalk();

}

}

}

}

Так как же работает очередь? Всего существует 3 переменные ответственные за неё: **queue**, **queueFood**, **queueWood**. “queue” отвечает за очередь из цветов фракции, то есть это список цветов. “queueFood” отвечает за очередь банка еды, а “queueWood” за очередь банка ресурсов. Все необходимые элементы записаны под одним индексом.

Значит очередь в общем выглядит так, по порядку:

Индекс 0 1 2

queue [ (цвет фракции 1), (цвет фракции 2), (цвет фракции 3) ]

queueFood [ (банк еды фракции 1), (банк еды фракции 2), (банк еды фракции 3) ]

queueWood [ (банк дерева фракции 1), (банк дерева фракции 2), (банк дерева фракции 3) ]

#### Экономика

Время экономики. Так как я представляю скрипты итогового проекта, вы уже видели её готовые элементы. За экономику отвечает в основном my\_events и в нём есть две переменные: rewardFood и rewardWood. Они отвечают за награду получаемую фракцией при смене хода, причём той что ходила, а не той, что будет ходить. За их значения отвечает следующий метод:

public void Update() // Расчёт происходит каждый кадр

{

rewardWood = 0;

rewardFood = 0;

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

foreach (GameObject box in boxes)

{

if (box.GetComponent<Renderer>().material.color == turn) // Расчёт происходит только для блоков цвета фракции хода

{

rewardFood += rewardFoodForBlock;

rewardWood += rewardWoodForBlock;

if (box.GetComponent<box>().economic\_zone == "forest")

{

rewardWood += rewardForForest;

}

if (box.GetComponent<box>().economic\_zone == "animals")

{

rewardFood += rewardForAnimals;

}

}

}

GameObject[] characters = GameObject.FindGameObjectsWithTag("character");

foreach (GameObject character in characters)

{

if (character.GetComponent<character>().colorful\_details.GetComponent<Renderer>().material.color == turn) // Расчёт происходит только для персонажей цвета фракции хода

{

{

rewardFood -= sanctionForUnit \* character.GetComponent<character>().power + ( 2 \* (character.GetComponent<character>().power-1) \* sanctionForUnit);

}

}

GameObject[] selected = GameObject.FindGameObjectsWithTag("selected");

foreach (GameObject select in selected) // Расчёт включает и выделенных персонажей фракции

{

rewardFood -= sanctionForUnit \* select.GetComponent<character>().power + (2 \* (select.GetComponent<character>().power - 1) \* sanctionForUnit);

}

if (rewardFood>0)

{

foodRewardText.text = "+" + rewardFood.ToString();

} else

{

foodRewardText.text = rewardFood.ToString();

}

if (rewardWood > 0)

{

woodRewardText.text = "+" + rewardWood.ToString();

}

else

{

woodRewardText.text = rewardWood.ToString();

}

}

Здесь мы можем увидеть что есть 3 составляющие экономики:

1. Заработок с территорий
2. Заработок с экономических зон
3. Издержки армии

И их реализацию:

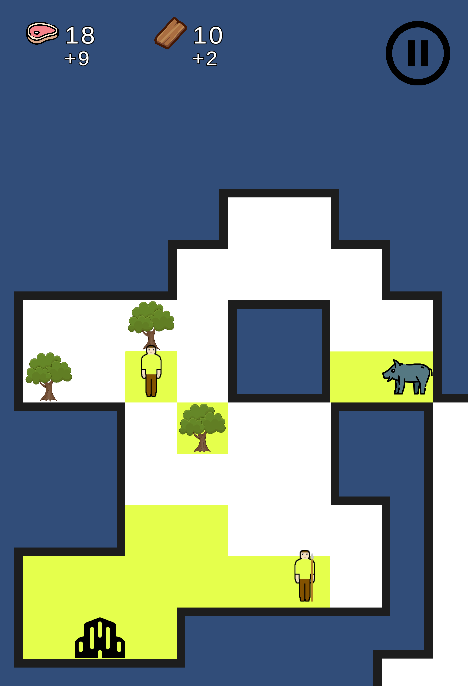
1. Награду получаемую с каждого блока фракции представляют переменные **rewardFoodForBlock** и **rewardWoodForBlock**. Для них установлены значения 1 на еду и 0 на дерево.
2. Экономическая зона (economic\_zone) – это зоны с повышенной наградой. Переменные **rewardForAnimals** и **rewardForForest** ответственны за прибавку с экономической зоны. В скрипте у обоих установлено значение 1, но в юнити это значение изменено на: для леса – 2, а для животных – 4.
3. За издержки армии отвечает переменная **sanctionForUnit**. Расчёт долга прост: это сумма всех издержек юнитов расчитанных по следующей формуле: (сила персонажа) \* sanctionForUnit

То есть награда будет вычисляться так:

(награда за блок) + (награда за экономическую зону) – (издержки армии).

Значит на экономической зоне с деревом вы будете получать не 0, а 2 дерева за ход, а на зоне с животными не 1, а 5 еды.

Узнать сколько вы получите ресурсов под конец хода, а также сколько вы потратите на приобретение очередного игрового персонажа, можно узнать из элементов игрового интерфейса:

1)  2) 

*\*1 банк валют \*2 магазин*

За генерацию экономических придерживается следующих правил:

1. Экономические зоны не должны быть на территории фракций
2. Экономические зоны не могут быть инициированы в определённом радиусе от главных зданий
3. Шанс генерации леса – 9%, а животных 1%

Вот часть кода генератора описывающая экономические зоны:

…

boxes = GameObject.FindGameObjectsWithTag("box");

foreach (GameObject box in boxes)

{

if (box.GetComponent<box>().building == null)

{

bool availableToZone = false;

foreach (GameObject faction in factions)

{

if ((Vector3.Distance(box.transform.position, faction.transform.position) > 1) & (box.GetComponent<Renderer>().material.color == new Color(1, 1, 1, 1)))

{

availableToZone = true;

}

else

{

availableToZone = false;

break;

}

}

if (availableToZone)

{

int chance = rnd.Next(1, 101);

if (chance >= 100)

{

box.GetComponent<box>().economic\_zone = "animals";

Instantiate(animals, box.transform.position, box.transform.rotation).transform.parent = box.transform;

}

else if (chance >= 92)

{

box.GetComponent<box>().economic\_zone = "forest";

Instantiate(forest, box.transform.position, box.transform.rotation).transform.parent = box.transform;

}

}

}

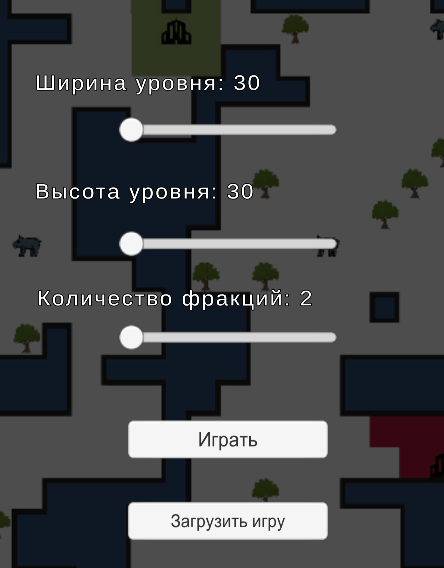
}

Для найма, каждый юнит требует определённое количество еды, а здания и особо сильные юниты ещё и дерево. Так как дерево можно получить только с экономической зоны – это стратегически важный ресурс, как и животные, которые дают заметное преимущество в еде, особенно в начале игры.

### Переходы между сценами и их взаимодействие

#### Главное меню

Так, у нас есть геймплей. Самое время для не мало важной части каждой игры – главное меню.

Создадим интерфейс для него:

В интерфейсе главного меню мы можем настраивать уровень по длине и ширине, а также выбрать количество фракций. Максимальное количество фракций будет зависеть от размеров уровня.

Напишем скрипт и наложим его на кнопку “играть” или же “NEW GAME”. Получается вот так:

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.SceneManagement;

public class Options : MonoBehaviour

{

// Update is called once per frame

public string saveFile;

void Start()

{

Application.targetFrameRate = 30;

saveFile = Application.persistentDataPath + "/gamedata.txt";

if (!File.Exists(saveFile))

{

GameObject.Find("LOAD").GetComponent<Button>().interactable = false;

} else

{

GameObject.Find("LOAD").GetComponent<Button>().interactable = true;

}

}

public TextMeshProUGUI width\_text;

public TextMeshProUGUI height\_text;

public TextMeshProUGUI factionCount\_text;

public float value\_width; // Значение в ползунке ширины

public double convert\_value\_width; // Ширина в блоках

public float value\_height; // Значение в ползунке длинны

public double convert\_value\_height; // Длинна в блоках

public float value\_factionCount; // Значение в ползунке фракций

public double MaxFactionCount;

public double convert\_factionCount; // Количество фракций

public bool loadLevel = false;

public bool contUpdate = true;

void Update()

{

if (contUpdate)

{

//width

value\_width = GameObject.Find("WIDTHLEVEL").GetComponent<Slider>().value;

convert\_value\_width = 15 + (Math.Round(35 \* value\_width, 0));

width\_text.text = "Ширина уровня: " + (convert\_value\_width \* 2).ToString();

//height

value\_height = GameObject.Find("HEIGHTLEVEL").GetComponent<Slider>().value;

convert\_value\_height = 15 + (Math.Round(35 \* value\_height, 0));

height\_text.text = "Высота уровня: " + (convert\_value\_height \* 2).ToString();

//faction

value\_factionCount = GameObject.Find("FACTIONSCOUNT").GetComponent<Slider>().value;

MaxFactionCount = 2 + Math.Round((convert\_value\_width \* convert\_value\_height) / 300);

convert\_factionCount = 2 + (Math.Round((MaxFactionCount - 2) \* value\_factionCount, 0));

factionCount\_text.text = "Количество фракций: " + convert\_factionCount.ToString();

//write in tempclass

TempClass.set((float)convert\_value\_width, (float)convert\_value\_height, Convert.ToInt32(convert\_factionCount), loadLevel);

} else

{

TempClass.set((float)convert\_value\_width, (float)convert\_value\_height, Convert.ToInt32(convert\_factionCount), true);

SceneManager.LoadScene("Game");

}

}

public void ButtonEvent()

{

SceneManager.LoadScene("Game");

}

public void ButtonLoadEvent()

{

contUpdate = false;

}

}

Очень важный элемент, который создаёт связь между сценами главного меню и игрового процесса – это “TempClass” (Временный класс). Юнити хорошо умеет работать со скриптами, даже с теми, что не принадлежат ни одному объекту. Этим я и воспользовался, когда присвоил значения из настроек главного меню временному классу.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class TempClass

{

private static float value\_width = 20;

private static float value\_height = 20;

private static int factionCount = 2;

private static bool loadLevel = true;

public static void set(float convert\_value\_width, float convert\_value\_height, int convert\_factionCount, bool convert\_loadLevel)

{

value\_width = convert\_value\_width;

value\_height = convert\_value\_height;

factionCount = convert\_factionCount;

loadLevel = convert\_loadLevel;

}

public static bool get\_loadLevel()

{

return loadLevel;

}

public static float get\_width()

{

return value\_width;

}

public static float get\_height()

{

return value\_height;

}

public static int get\_factionCount()

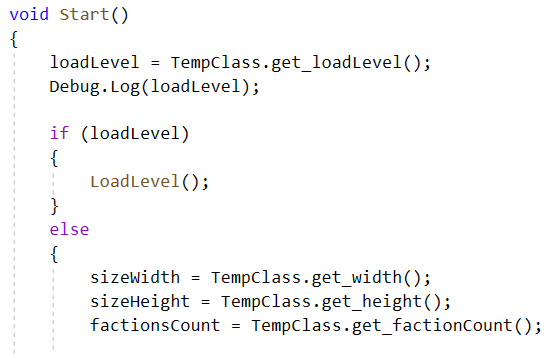
{

return factionCount;

}

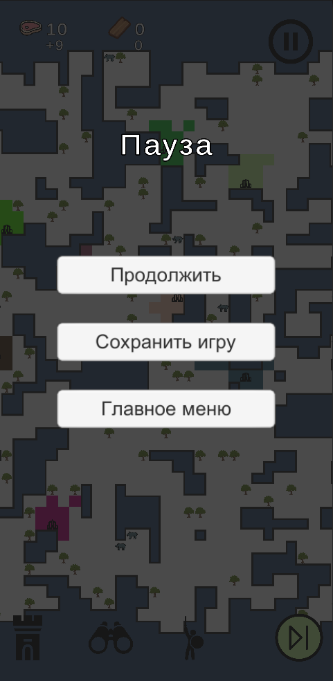
}

Эти значения берёт генератор уровня перед началом своей деятельности:



#### Меню паузы

Главное меню – есть, переход из игры в главное меню – нет, надо сделать:

Меню паузы состоит из трёх кнопок, заднего фона и текста “пауза”. Все элементы паузы связанны позицией с задним фоном. Задний фон представляет из себя серую полупрозрачную текстуру, растянутую во весь экран. Кнопка “продолжить” уводит меню за видимость камеры, кнопка “сохранить игру” сохраняет уровень, “главное меню” – переключает сцену на главное меню.

Пока открыта пауза, с элементами игры, находящимися на заднем фоне, невозможно взаимодействовать.

Анимация появления представляет из себя перемещение невидимого меню, а после его проявления.

Скрипт, описывающий взаимодействие к кнопками наложен за задний фон меню паузы, кроме кнопки “сохранить игру”, которая использует метод SaveLevel из класса “my\_events”.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.SceneManagement;

public class PauseMenuScript : MonoBehaviour

{

bool call = false; // Вызвана ли пауза

float step = 0; // Кадр анимации

float stepCount = 10; // Количество кадров в анимации

public void GotoMainMenu()

{

SceneManager.LoadScene("mainMenu");

}

public void CallPause()

{

transform.position = new Vector3(transform.position.x, transform.position.y - (1000 \* Screen.height), transform.position.z);

call = true;

Camera.main.GetComponent<CamEvents>().pause = true;

}

public void ClosePause()

{

step = 0;

transform.position = new Vector3(transform.position.x, transform.position.y + (1000 \* Screen.height), transform.position.z);

Color myColor = gameObject.GetComponent<Image>().color;

gameObject.GetComponent<Image>().color = new Color(myColor.r, myColor.g, myColor.b, 0);

foreach (Transform t in transform)

{

if (t.gameObject.name != "pauseText (TMP)")

{

Color tColor = t.gameObject.GetComponent<Image>().color;

t.gameObject.GetComponent<Image>().color = new Color(tColor.r, tColor.g, tColor.b, 0);

Transform tText = t.Find("text");

if (tText != null)

{

tColor = tText.gameObject.GetComponent<Text>().color;

tText.gameObject.GetComponent<Text>().color = new Color(tColor.r, tColor.g, tColor.b, 0);

}

}

else

{

Color tColor = t.gameObject.GetComponent<TextMeshProUGUI>().color;

t.gameObject.GetComponent<TextMeshProUGUI>().color = new Color(tColor.r, tColor.g, tColor.b, 0);

}

}

Camera.main.GetComponent<CamEvents>().pause = false;

}

void Update() // Анимация появления

{

if (call)

{

step++;

Color myColor = gameObject.GetComponent<Image>().color;

gameObject.GetComponent<Image>().color = new Color(myColor.r, myColor.g, myColor.b, 0.8f \* (step / stepCount));

foreach (Transform t in transform)

{

if (t.gameObject.name != "pauseText (TMP)")

{

Color tColor = t.gameObject.GetComponent<Image>().color;

t.gameObject.GetComponent<Image>().color = new Color(tColor.r, tColor.g, tColor.b, 1 \* (step / stepCount));

Transform tText = t.Find("Text");

if (tText != null)

{

tColor = tText.gameObject.GetComponent<Text>().color;

tText.gameObject.GetComponent<Text>().color = new Color(tColor.r, tColor.g, tColor.b, 1 \* (step / stepCount));

}

}

else

{

Color tColor = t.gameObject.GetComponent<TextMeshProUGUI>().color;

t.gameObject.GetComponent<TextMeshProUGUI>().color = new Color(tColor.r, tColor.g, tColor.b, 1 \* (step / stepCount));

}

}

if (step == stepCount)

{

call = false;

}

}

}

}

### Сохранения

Вот и готова игры, но не хватает лишь одной, великой и прекрасной детали – сохранений. Не всегда возможно доиграть эпическую бойню, начатую с другом 10 минут назад, необходимо сохранять игровой процесс, чтобы не тратить попусту мощности телефона, на постоянный расчёт графики или кода.

Но как написать сохранения? Просто! Есть два способа:

1. Юнити предлагает записывать данные игр в регистр, как это делают и другие игры, записывая айди сохранённой игры.
2. Json файл. Записать все необходимые данные в него. В отличие от первого способа, в json можно записать класс, а не просто переменную.

Решено. Мой выбор json.

Как можно представить мой уровень в файле json? Ответ прост, в виде класса. Готовая игра имеет завязку на блоках. К блоку привязаны персонажи и здания, на блоках завязаны фракции. Цвета игровых объектов не прямо зависимы от блоков, на которых они находятся, но они всегда совпадают. Координаты блоков представленные классом Vector3, можно записать в виде координат x, y, z, как и цвета, представленные классом Color, можно записать в формате rgba. Пусть у каждого игрового объекта будет свой айди и если юнит может походить, то запишем это. Не за будет про экономические зоны и получается следующий класс:

[System.Serializable] // Показывает, кто класс можно сериализовать (получить из json объекта)

public class GameData

{

public float[] position = new float[3]; // позиция xyz

public float[] col = new float[4]; // цвет rgba

public int buildingID;

public int characterID;

public bool characterCanWalk;

public string economicZone;

public void setValue(GameData gameData)

{

position = gameData.position;

col = gameData.col;

buildingID = gameData.buildingID;

characterID = gameData.characterID;

economicZone = gameData.economicZone;

}

}

Отлично, но игра у нас пошаговая, а значит нужно добавить ещё и класс ответственный за очередь, банки валют и фракцию ходящую в данный момент. Вот он:

[System.Serializable]

public class QueueData

{

public float[] queue; // Массив очереди. Включает все элементы очереди

public float sizeWidth; // Ширина уровня

public float sizeHeight; // Высота уровня

public void setValue(QueueData queueData)

{

sizeWidth = queueData.sizeWidth;

sizeHeight = queueData.sizeHeight;

queue = queueData.queue;

}

}

Данные классы находятся в генераторе уровня.

Так как мы работаем с json, необходимо создать инструменты для этого. Юнити даёт нам записать класс в виде json объекта, но только одни класс, а мне нужно записать множество классов и причём одни из них будет другим. Используя особенности сериализации юнити, я создал это:

public class GameDataManager : MonoBehaviour

{

// Create a field for the save file.

public string saveFile;

public string[] readFile()

{

saveFile = Application.persistentDataPath + "/gamedata.txt"; // Путь сохранения

Debug.Log(saveFile);

// Does the file exist?

if (File.Exists(saveFile))

{

// Read the entire file and save its contents.

string fileContents = File.ReadAllText(saveFile);

string[] fileJson = fileContents.Split(new char[] { '|' }, StringSplitOptions.RemoveEmptyEntries); // Разбить текст на массив, где разделение служит символ “|”

Debug.Log(fileJson.Length);

return fileJson; // Вернуть массив со строками json

}

return null;

}

public void writeFile(float sizeWidth, float sizeHeight)

{

GameObject[] boxes = GameObject.FindGameObjectsWithTag("box");

string toSave = null; // Строка, которая будет записана в файл txt

for (int i = 0; i < boxes.Length; i++)

{

// Запись свойств в класс GameData

GameData gameData = new GameData();

gameData.position = new float[] { boxes[i].transform.position.x, boxes[i].transform.position.y, boxes[i].transform.position.z };

gameData.col = new float[] { boxes[i].GetComponent<Renderer>().material.color.r, boxes[i].GetComponent<Renderer>().material.color.g, boxes[i].GetComponent<Renderer>().material.color.b, boxes[i].GetComponent<Renderer>().material.color.a };

if (boxes[i].GetComponent<box>().building != null)

{

gameData.buildingID = boxes[i].GetComponent<box>().building.GetComponent<BuildingScript>().id;

}

else if (boxes[i].GetComponent<box>().character != null)

{

gameData.characterID = boxes[i].GetComponent<box>().character.GetComponent<character>().id;

gameData.characterCanWalk = boxes[i].GetComponent<box>().character.GetComponent<character>().CanWalk;

}

gameData.economicZone = boxes[i].GetComponent<box>().economic\_zone;

// Запись строки в формате (json GameData)|

toSave += JsonUtility.ToJson(gameData) + "|";

}

// Запись свойств в класс QueueData

QueueData queueData = new QueueData();

my\_events my\_events = GameObject.FindGameObjectsWithTag("my\_events")[0].GetComponent<my\_events>();

float[] queue = my\_events.get\_queue();

queueData.queue = queue;

queueData.sizeWidth = sizeWidth;

queueData.sizeHeight = sizeHeight;

// Запись класса QueueData в строку в json предствалении

toSave += JsonUtility.ToJson(queueData);

saveFile = Application.persistentDataPath + "/gamedata.txt"; // Путь сохранения

Debug.Log(saveFile);

// Serialize the object into JSON and save string.

// Write JSON to file.

File.WriteAllText(saveFile, toSave); // Запись строки toSave в файл saveFile

}

}

Здесь заметен новый метод класса my\_events: get\_queue

public float[] get\_queue()

{

float[] result = new float[(queue.Count \* 6)+1];

for (int i=0; i<queue.Count; i+=1)

{

result[i+ (i \* 5)] = queue[i].r;

result[i+ (i \* 5) + 1] = queue[i].g;

result[i+ (i \* 5) + 2] = queue[i].b;

result[i+ (i \* 5) + 3] = queue[i].a;

result[i+ (i \* 5) + 4] = (float)queueFood[i];

result[i+ (i \* 5) + 5] = (float)queueWood[i];

}

result[queue.Count \* 6] = (float)ind;

return result;

}

“get\_queue” отвечает за создание массива очереди. Каждая фракция, банк валют в массиве записаны и идентичном порядке очереди, как и было изначально. В формате:

(Цвет r,g,b,a), (Банк еды), (Банк ресурсов)…(номер фракции, которой принадлежит ход)

Пример для четырёх фракций:

[**0.800000011920929**,**1.0**,**0.6000000238418579**,**1.0**,*10.0*,*0.0*,**0.0**,**0.20000000298023225**,**0.6000000238418579**,**1.0**,*10.0*,*0.0*,**0.10000000149011612**,**0.800000011920929**,**0.20000000298023225**,**1.0**,*10.0*,*0.0*,**0.30000001192092898**,**0.20000000298023225**,**0.10000000149011612**,**1.0**,*10.0*,*0.0*,**0.800000011920929**,**0.0**,**0.5**,**1.0**,*10.0*,*0.0*,0.0]

**Жирным** – выделены цвета rgba

*Курсивом* – банк валют в порядке: еда, дерево

Подчёркиванием – номер фракции, которой принадлежит ход

Класс GameDataManager находится в скрипте генератора уровня, как и ранее упомянутые классы записываемые в json.

Обращение к этому классу исходят из класса levelGenerator, от методов LoadLevel и SaveLevel. LoadLevel я покажу позже, а SaveLevel выглядит так:

public void SaveLevel()

{

GameDataManager gameDataManager = new GameDataManager();

gameDataManager.writeFile(sizeWidth, sizeHeight);

Debug.Log("save ready");

}

SaveLevel использует кнопка сохранения, находящаяся в меню паузы.

Итоговый файл, сохраняемый методом “writeFile”, является txt форматом, но внутри него записаны строчки json. В общем получается структура:

**(json GameData)|(json GameData)|…|(json QueueData)**

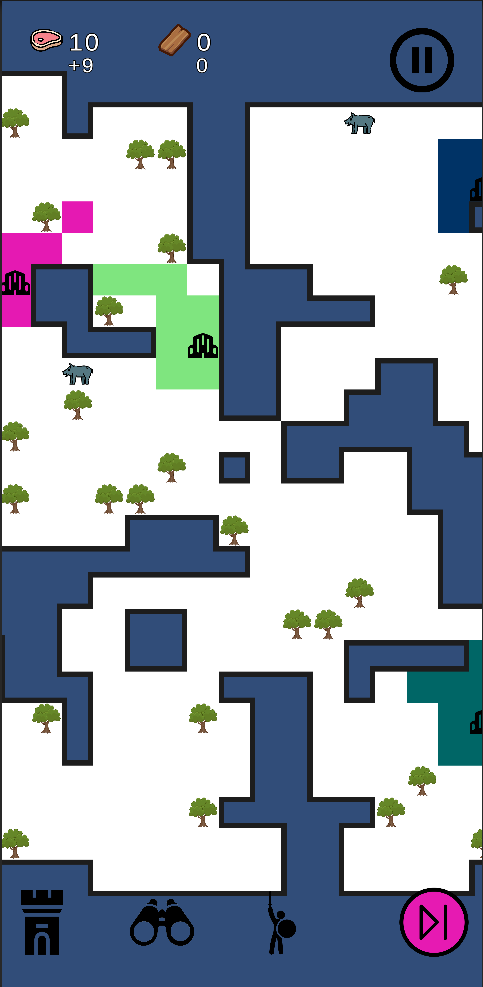
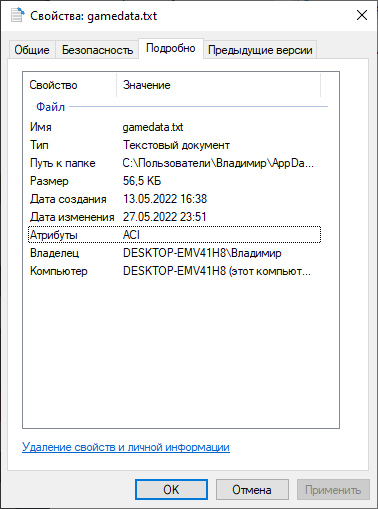
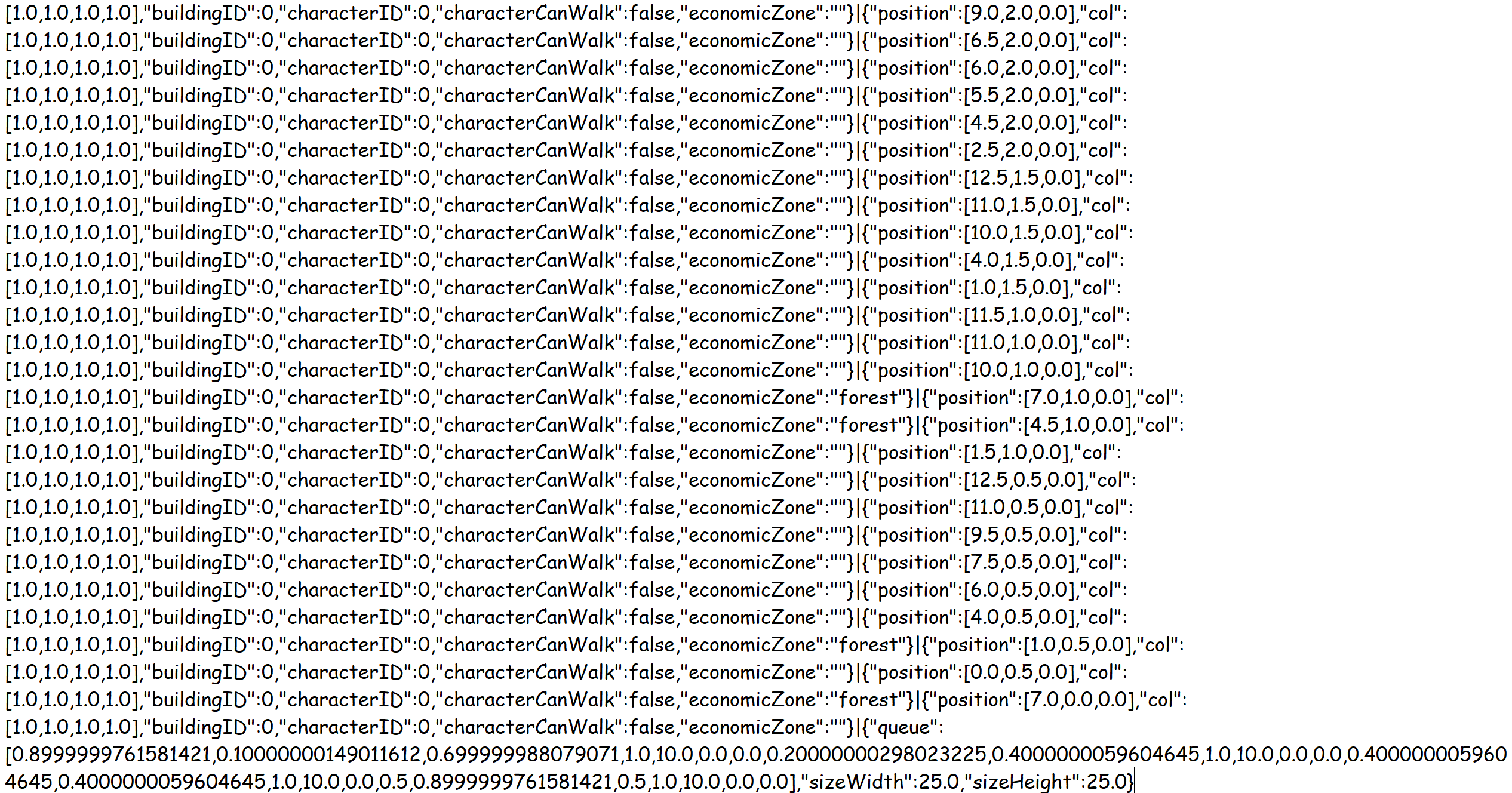
Пример GameData:

{"position":[3.0,2.5,0.0],"col":[1.0,1.0,1.0,1.0],"buildingID":0,"characterID":0,"characterCanWalk":false,"economicZone":""}

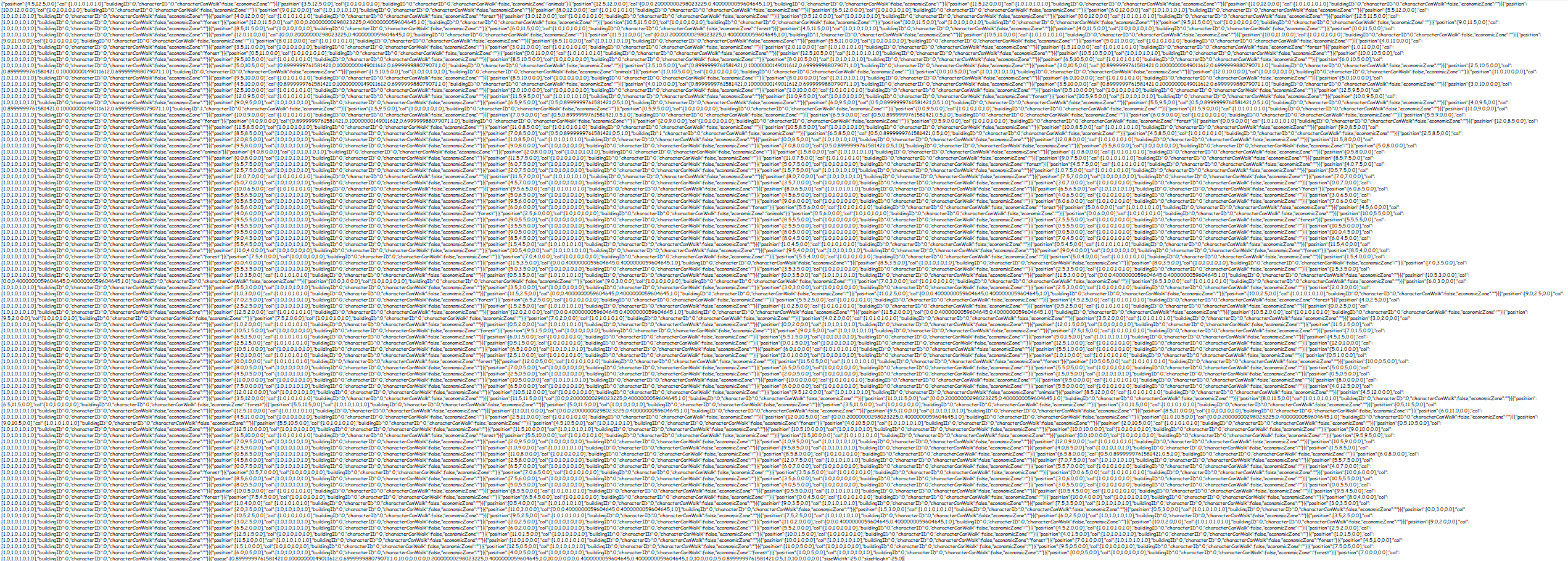
Пример QueueData:

{"queue":[0.0,0.800000011920929,0.30000001192092898,1.0,10.0,0.0,0.4000000059604645,0.20000000298023225,0.30000001192092898,1.0,10.0,0.0,0.0],"sizeWidth":15.0,"sizeHeight":15.0}

Готовое сохранение уровня 40 на 40, с 4 фракциями выглядит так:

1)  2)  3) 

*\*1 уровень, \*2 свойства файла, \*3 часть файла*



*\*Общий вид файла*

Мы научились записывать сцену в json файл. Осталось лишь её прочитать. Метод readFile даёт нам массив всех json строк и мы знаем, что все кроме последней строки – классы GameData, последняя же – класс QueueData.

Загрузка уровня происходит следующим образом:

public void LoadLevel()

{

GameDataManager gameDataManager = new GameDataManager();

string[] fileJson = gameDataManager.readFile();

QueueData ResultQueue = new QueueData();

ResultQueue.setValue(JsonUtility.FromJson<QueueData>(fileJson[fileJson.Length - 1])); // Десереализация класса QueueData

// Десереализация классов GameData

for (int i = 0; i < fileJson.Length - 1; i++)

{

GameData gameData = JsonUtility.FromJson<GameData>(fileJson[i]);

// Спавн блоков нужно цвета

GameObject oneOfBox = Instantiate(box, new Vector3(gameData.position[0], gameData.position[1], gameData.position[2]), box.transform.rotation);

oneOfBox.GetComponent<Renderer>().material.color = new Color(gameData.col[0], gameData.col[1], gameData.col[2], gameData.col[3]);

// Инициализация экономических зон

if (gameData.economicZone != "")

{

oneOfBox.GetComponent<box>().economic\_zone = gameData.economicZone;

if (gameData.economicZone == "forest")

{

Instantiate(forest, new Vector3(gameData.position[0], gameData.position[1], gameData.position[2]), box.transform.rotation);

}

else if (gameData.economicZone == "animals")

{

Instantiate(animals, new Vector3(gameData.position[0], gameData.position[1], gameData.position[2]), box.transform.rotation);

}

}

// Инициализация персонажей

if (gameData.characterID != 0)

{

foreach (GameObject character in unitList)

{

if (character.GetComponent<character>().id == gameData.characterID)

{

GameObject spawnCharacter = Instantiate(character, new Vector3(gameData.position[0], gameData.position[1], gameData.position[2]), box.transform.rotation);

spawnCharacter.GetComponent<character>().SetColor(new Color(gameData.col[0], gameData.col[1], gameData.col[2], gameData.col[3]));

spawnCharacter.GetComponent<character>().block = oneOfBox;

if (!gameData.characterCanWalk)

{

spawnCharacter.GetComponent<character>().SetNotWalk();

}

break;

}

}

}

// Инициализация строений

else if (gameData.buildingID != 0)

{

foreach (GameObject building in buildingList)

{

if (building.GetComponent<BuildingScript>().id == gameData.buildingID)

{

GameObject spawnBuilding = Instantiate(building, new Vector3(gameData.position[0], gameData.position[1], gameData.position[2]), box.transform.rotation);

spawnBuilding.GetComponent<BuildingScript>().SetColor(new Color(gameData.col[0], gameData.col[1], gameData.col[2], gameData.col[3]));

spawnBuilding.GetComponent<BuildingScript>().block = oneOfBox;

break;

}

}

}

}

// Центрирование камеры

sizeWidth = ResultQueue.sizeWidth;

sizeHeight = ResultQueue.sizeHeight;

float sizeWidthTemp = sizeWidth \* distance;

float sizeHeightTemp = sizeHeight \* distance;

Camera.main.GetComponent<CamEvents>().sizeWidthTemp = sizeWidthTemp;

Camera.main.GetComponent<CamEvents>().sizeHeightTemp = sizeHeightTemp;

float max = Math.Max(sizeWidth, sizeHeight);

Camera.main.transform.position = new Vector3(sizeWidthTemp / 2, sizeHeightTemp / 2, -10);

if (camSet)

{

Camera.main.GetComponent<Camera>().orthographicSize = max \* (0.33f - (max / 10 \* 0.006f));

Camera.main.GetComponent<CamEvents>().zoomMax = max \* (0.33f - (max / 10 \* 0.006f));

}

// Запуск игры

my\_events my\_events = GameObject.FindGameObjectsWithTag("my\_events")[0].GetComponent<my\_events>();

my\_events.load\_game(ResultQueue.queue);

}

Метод load\_game в my\_events (похож на start\_game):

public void load\_game(float[] receivedQueue)

{

Debug.Log("Screen size:");

Debug.Log("==============");

Debug.Log("width:");

Debug.Log(Screen.width);

Debug.Log("height:");

Debug.Log(Screen.height);

Debug.Log("==============");

if (Screen.height >= 1280)

{

GameObject.FindGameObjectsWithTag("animation helper")[0].transform.position = new Vector3((Screen.width \* 6) / 12, ((Screen.height \* 2.5f) / 15) - 1000 \* (Screen.height / 1280), 0);

}

else

{

GameObject.FindGameObjectsWithTag("animation helper")[0].transform.position = new Vector3((Screen.width \* 6) / 12, ((Screen.height \* 2.5f) / 15) - 1000 \* (Screen.height \* 0.00078125f), 0);

}

PauseBackground.transform.position = new Vector3(PauseBackground.transform.position.x, PauseBackground.transform.position.y + (1000 \* Screen.height), PauseBackground.transform.position.z);

GameObject PauseButton = GameObject.FindGameObjectsWithTag("PauseButton")[0];

PauseButton.transform.position = new Vector3((Screen.width \* 10.5f) / 12, (Screen.height \* 14.1f) / 15, PauseButton.transform.position.z);

GameObject UnitButton = GameObject.FindGameObjectsWithTag("UnitButton")[0];

UnitButton.transform.position = new Vector3((Screen.width \* 7) / 12, Screen.height / 15, UnitButton.transform.position.z);

GameObject ColonistButton = GameObject.FindGameObjectsWithTag("ColonistButton")[0];

ColonistButton.transform.position = new Vector3((Screen.width \* 4) / 12, Screen.height / 15, ColonistButton.transform.position.z);

GameObject BuildingButton = GameObject.FindGameObjectsWithTag("BuildingButton")[0];

BuildingButton.transform.position = new Vector3((Screen.width \* 1) / 12, Screen.height / 15, BuildingButton.transform.position.z);

GameObject NextTurnButton = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

NextTurnButton.transform.position = new Vector3((Screen.width \* 10.8f) / 12, Screen.height / 15, NextTurnButton.transform.position.z);

GameObject foodGameO = GameObject.FindGameObjectsWithTag("food")[0];

foodGameO.transform.position = new Vector3((Screen.width \* 2.9f) / 12, (Screen.height \* 14.4f) / 15, food.transform.position.z);

GameObject woodGameO = GameObject.FindGameObjectsWithTag("wood")[0];

woodGameO.transform.position = new Vector3((Screen.width \* 6.1f) / 12, (Screen.height \* 14.4f) / 15, food.transform.position.z);

for (int i = 0; i < receivedQueue.Length-1; i += 6)

{

queue.Add( new Color(receivedQueue[i], receivedQueue[i + 1], receivedQueue[i + 2], receivedQueue[i + 3]) );

queueFood.Add( (int)receivedQueue[i + 4] );

queueWood.Add( (int)receivedQueue[i + 5] );

}

ind = (int)receivedQueue[receivedQueue.Length - 1];

Debug.Log(ind);

turn = queue[ind];

food.text = queueFood[ind].ToString();

wood.text = queueWood[ind].ToString();

GameObject but = GameObject.FindGameObjectsWithTag("NextTurnButton")[0];

but.GetComponent<Image>().color = turn;

}

# ЗАКЛЮЧЕНИЕ

## 3.1. Баги и неполадки

Несмотря на попытки написать код без неполадок, без них не обошлось:

1. Баг с передвижение персонажей на неограниченное расстояние. Это связанно с тем, что несмотря как далека находится персонаж от блока, если этот блок не является территорией его фракции, но является касательным к такому, то персонаж может на него походить. Это можно исправить в функции SeeZone, класса character, внеся несколько изменений.

2. Ужасная оптимизация на больших уровнях. Код усыпан циклами и самые опасные из них – это циклы ищущие ближайшие блоки к объекту. Если этих блоков 400, то устройство вполне стравляется, но если их 10000, то здесь либо игра зависнет, либо произойдёт вылет. Этого можно было избежать в самом начале проекта, но у меня не получилось реализовать эту механику по-другому и поэтому, следуя своей философии при разработке, этот неприятный момент распространился на большую часть кода. Поэтому я ограничил максимальный размер уровня до 100\*100, но как я описал ранее, это всё равно слишком много.

## 3.2. Вывод

Этот проект для диплома я писал на протяжении месяца, а если брать весь срок (от написания первых строк до последней), то три месяца. За это время я разобрался в движке Unity, понял как работать C# хотя бы на уровне новичка, как работать с классами и объектами, как записывать информацию в текстовый и json файл и читать её. Конечно хотелось бы более развить некоторые механики, может быть я эти займусь в будущем.

# 4. СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

<https://docs.unity3d.com/ru/530/Manual/DrawCallBatching.htmlю>

<https://ru.wikipedia.org/wiki/Unity_(игровой_движок)>

<https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio>

<https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html>

<https://ru.wikipedia.org/wiki/Adobe_Photoshop>