# 3. Экспериментальная проверка разработанного алгоритма

# 3.1 Организация вычислительного эксперимента

# 3.1.1 Проектирование и разработка модели в Unity

Реализация алгоритма включает использование датчиков изменения высоты, которые располагаются на определенном расстоянии друг от друга. Сигналы от этих датчиков передаются на центральный компьютер или контроллер для обработки данных.

Для эмуляции функциональности устройства был выбран игровой движок Unity, который является популярным и предоставляет возможность создания виртуальных сред и моделирования физических процессов. В рамках Unity была разработана симуляция, которая имитировала считывание показателей с датчиков. Таким образом, была осуществлена проверка алгоритма определения количества яиц без использования реальных яиц и физической модели. Геометрические показатели зависимости размера яиц от категории соответствует рисунку 1.

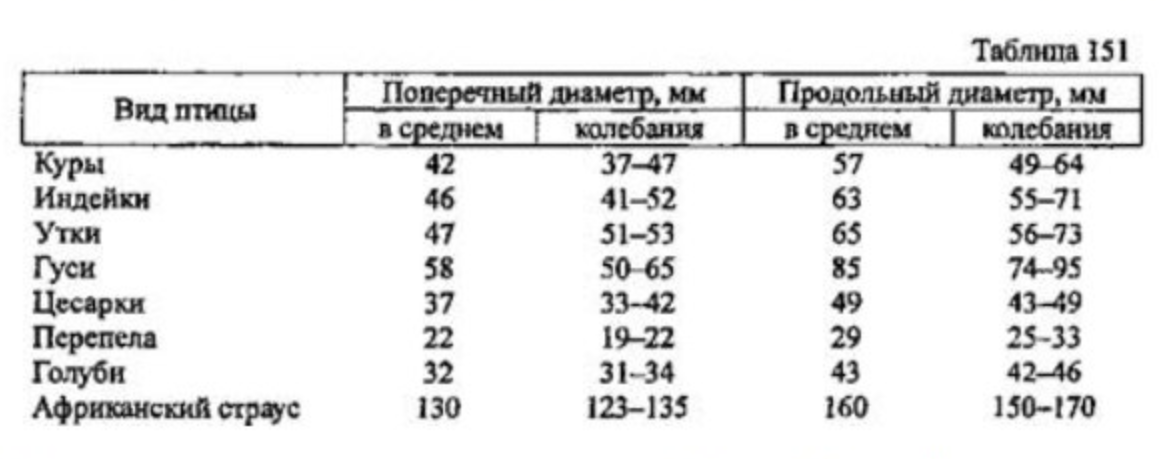


Рисунок 1 – Геометрические показатели зависимости размера яиц от категории

- генерирование яиц разных форм и позиций

На рисунке 2 представлена генерация объектов, учитывая различные категории и положение яиц.

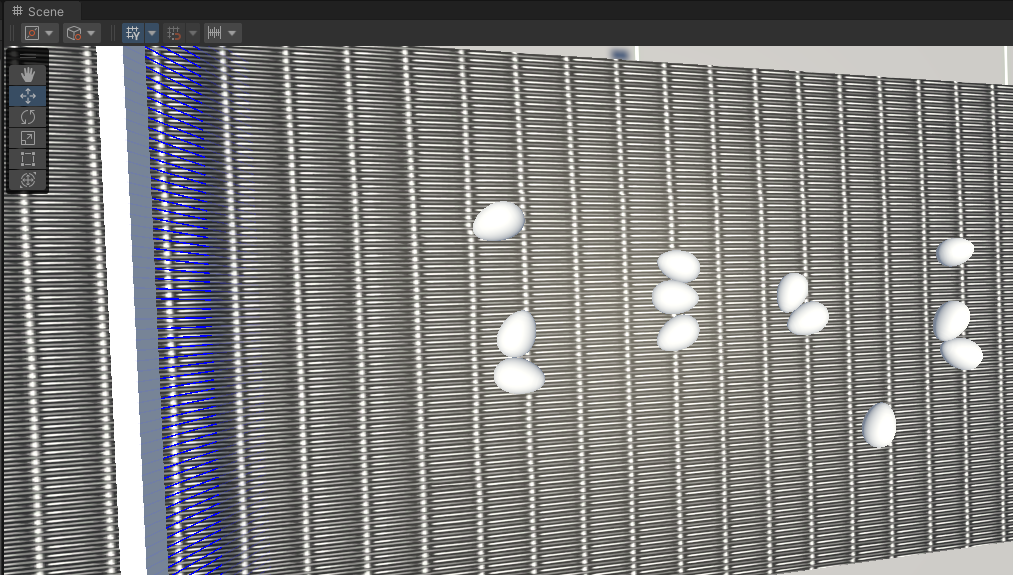


Рисунок 2 – Генерация яиц

Код генерации объектов представлен листингом 1.

Листинг 1 - Код генерации объектов

void Update()

{

float width = ConvWidthScript.getWidth();

if (Time.time > nextSpawn && GlobalVar.GetIsSpawn())

{

nextSpawn = Time.time + spawnDelay;

CalcEgg.addEggSpawnCol(spawnCol);

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

{

randomX = 0;

float rotateEgg = Random.Range(0.0f, 360.0f);

whereToSpawn = new Vector3(randomX, 2.0f, -7f);

GameObject Enemy = Instantiate(go, whereToSpawn, Quaternion.identity);

Enemy.transform.Rotate(0.0f, rotateEgg, 90f);

Enemy.transform.localScale = new Vector3(0.011f \* randomScale, 0.011f \* randomScale, 0.011f \* randomScale);//0.021

}

}

}

- модель датчика и как получаем показатели

В Unity роль датчиков выполняют примитивные кубы отправляют лучи (ray) через определенные промежутки времени. Основные характеристики шума и частоты обновления регулируются на основе настоящих датчиков VL53L0X. Внешний вид датчиков отображен на рисунке 3. Ширина датчиков, частота обновления и расстояние до конвейера могут быть настроены. Скорость движения конвейера можно задать в пределах от 1м/мин до 12 м/мин.

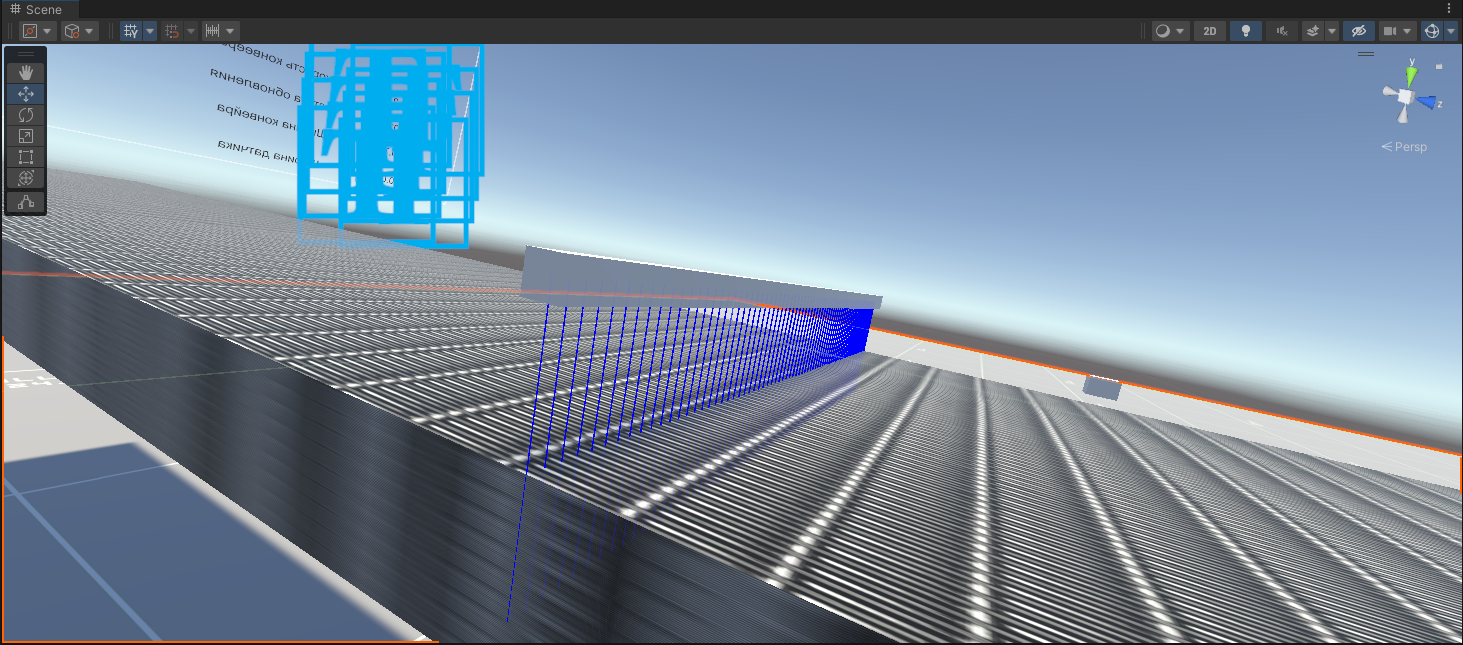


Рисунок 3 – Внешний вид датчиков

Код фиксирования показателей датчика представлен листингом 2.

Листинг 2 - Код фиксирования показателей датчика

void FixedUpdate()

{

nextUpdate = Time.time + GlobalVar.getSensorUpdateDelay();

Ray ray = new Ray(transform.position, transform.forward \* 0.15f);

Debug.DrawRay(transform.position, transform.forward \* 0.15f, Color.blue);

RaycastHit hit;

if (!Physics.Raycast(ray, out hit, 0.45f))

return;

float randomScale = Random.Range(-GlobalVar.getNoise(), GlobalVar.getNoise());

mas.Add(hit.distance + randomScale);

bool isIndexLast = (index == HeightMap.getMass().Count - 1);

if (isIndexLast)

{

HeightMap.CheckSensors();

}

if (hit.collider.gameObject.GetInstanceID() == -1268)

return;

float val = mas[mas.Count - 1];

if (val >= GlobalVar.getMinDistance())

return;

hit.collider.gameObject.GetComponent<Renderer>().material.color = Color.red;

Destroy(hit.collider.gameObject, 5f);

}

- сохранение данных и формирование бинарных файлов

Записанные показания с датчиков сохраняются в бинарный файл. Структура бинарного файла представлена на рисунке 4.

Рисунок 4 – Структура бинарного файла

Код записи в бинарный файл представлен листингом 3.

Листинг 3 - Код записи в бинарный файл

public static void WriteFile()

{

string path = "C:\\Users\\NIKITA-PC\\Desktop\\data\\file.bin";

using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))

{

float distanceToConv = 0.082f;

var outData = new OutData();

outData.count = heightMap.Count;

outData.iteration = heightMap[0].Count;

outData.col = CalcEgg.getEggSpawnCol();

outData.update = GlobalVar.getSensorUpdateDelay();

outData.width = ConvWidthScript.getWidth();

float distanceToConvNew = distanceToConv;

distanceToConvNew \*= 5;

distanceToConvNew \*= 65535;

distanceToConvNew = Math.Abs(distanceToConvNew - 65535);

outData.distanceToConv = (int)distanceToConvNew;

outData.noise = GlobalVar.getNoise();

var size = Marshal.SizeOf(outData);

var buffer = new byte[size];

var ptr = Marshal.AllocHGlobal(size);

Marshal.StructureToPtr(outData, ptr, true);

Marshal.Copy(ptr, buffer, 0, size);

Marshal.FreeHGlobal(ptr);

writer.Write(buffer);

for (int j = 0; j < heightMap[0].Count - 1; j++)

{

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

{

float value = heightMap[i][j];

value \*= 5;

value \*= 65535;

value = Math.Abs(value - 65535);

writer.Write((int)value);

}

}

writer.Close();

Debug.Log("File has been written");

}

}

Общий вид сцены представлен на рисунке 5.

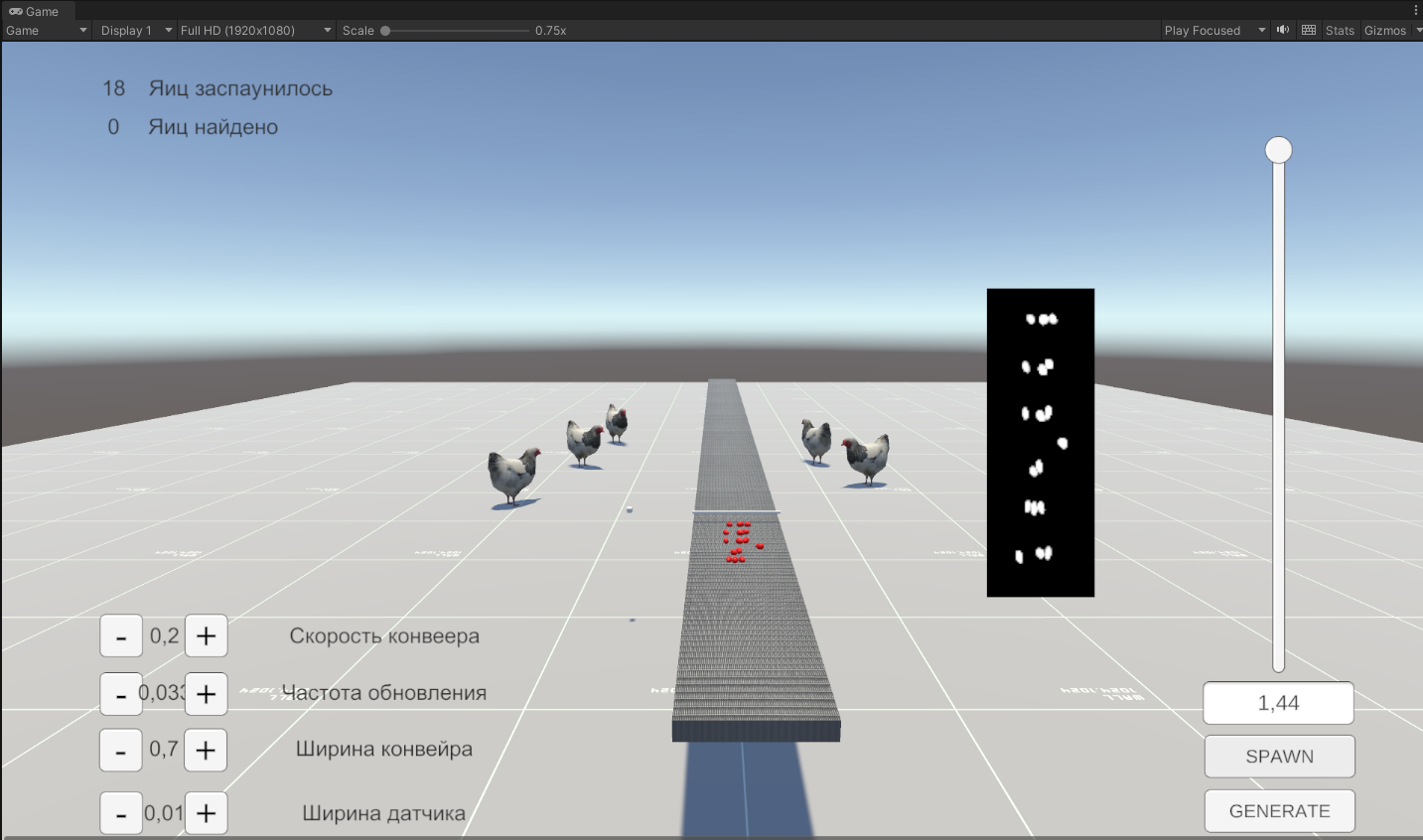


Рисунок 5 – Общий вид сцены

# 3.1.2 Проектирование и разработка приложения для оценки работы методов фильтрации

- чтение данных

- пример реализации выбранных

# 3.1.3 Проектирование и разработка приложения для оценки работы алгоритма

- чтение данных

- вывод изображения

- метод фильтрации

- все классы и методы

- небольшой результат

# 3.2 Анализ результатов эксперимента

# 3.3 Формирование рекомендаций

# 3.4 Выводы по главе.