АННОТАЦИЯ

В пояснительной записке представлено описание курсовой работы по дисциплине «Объектно-ориентированное программирование». Описан процесс разработки программы – компьютерной игры «Улей», имитирующей жизнедеятельность пчёл в дикой природе.

Документ содержит информацию о проектировании, создании и тестировании игры. Подробно представлена информация о процессах абстрагирования, выделения классов, проектирования системы, определения жизненного цикла программы и объектов в ней.

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ 4](#_Toc137409193)

[ВВЕДЕНИЕ 5](#_Toc137409194)

[СПИСОК ИСПОЛНИТЕЛЕЙ 6](#_Toc137409195)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#_Toc137409196)

[1.1 Цель разработки 7](#_Toc137409197)

[1.2 Описательная постановка задачи 7](#_Toc137409198)

[1.3 Условия выполнения и ограничения 8](#_Toc137409199)

[2 ПРОЕКТНОЕ РЕШЕНИЕ 9](#_Toc137409200)

[2.1 Абстрагирование и выделение объектов 9](#_Toc137409201)

[2.2 Построение иерархии классов 10](#_Toc137409202)

[2.3 Построение информационной модели 11](#_Toc137409203)

[2.4 Описание жизненного цикла программы 11](#_Toc137409204)

[2.5 Описание жизненного цикла одного из объектов 11](#_Toc137409205)

[2.6 Диаграмма переходов состояний 11](#_Toc137409206)

[2.7 Диаграмма потоков данных и действий 11](#_Toc137409207)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 12](#_Toc137409208)

[3.1 Обоснование выбора системы программирования 12](#_Toc137409209)

[3.2 Описание реализации основных классов и их методов 12](#_Toc137409210)

[3.3 Общее описание взаимодействия программных продуктов 13](#_Toc137409211)

[3.4 Интерфейс пользователя 14](#_Toc137409212)

[3.5 Критерии качества программной системы 16](#_Toc137409213)

[ЗАКЛЮЧЕНИЕ 17](#_Toc137409214)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 18](#_Toc137409215)

[ПРИЛОЖЕНИЕ А 19](#_Toc137409216)

[ПРИЛОЖЕНИЕ Б 22](#_Toc137409217)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ООП – объектно-ориентированное программирование.

IDE – англ. Integrated Development Environment, интегрированная среда разработки.

ОС – операционная система.

# ВВЕДЕНИЕ

Цель данного курсового проекта – закрепление знаний о технологии ООП, получение навыков разработки программ при поддержке объектно-ориентированного подхода.

Тема данного курсового проекта – проектирование и написание программы, реализующей основные принципы объектно-ориентированного программирования.

Для достижения поставленных целей были поставлены следующие задачи:

* выбор языка реализации, детализация постановки задачи;
* абстрагирование, разработка классов и их иерархии;
* написание текста программы на выбранном языке;
* разработка тестовых примеров;
* тестирование программы;
* разработка программных документов в соответствии с действующими стандартами.

По окончанию выполнения курсовой работы должны быть получены навыки решения задач объектно-ориентированным подходом.

# СПИСОК ИСПОЛНИТЕЛЕЙ

Текущая работа была выполнена студентами группы ИС/б-21-2-о:

1. Мельничук Владислав Викторович. Вклад: настройка игрового движка, написание кода для игровых объектов, разработка системы повышения уровней, разработка игрового интерфейса, дизайн двумерных моделей.
2. Шевелёв Кирилл Станиславович. Вклад: написание кода для игровых объектов, разработка системы вражеских волн, настройка игровой камеры, создание анимаций и музыкального сопровождения.
3. Мовенко Константин Михайлович. Вклад: написание кода для игровых объектов, разработка механик голода и регенерации, разработка боевого взаимодействия объектов, документирование.

Разработка идеи, определение классов, проектирование системы, настройка игрового баланса и тестирование осуществлялись коллективными усилиями.

# 1 ПОСТАНОВКА ЗАДАЧИ

## Цель разработки

Целью разработки является создание стратегической компьютерной игры «Улей», в которой игрок может управлять процессом жизнедеятельности диких пчёл.

## Описательная постановка задачи

Программа представляет из себя игру в жанре «Tycoon» - смесь симулятора и экономической стратегии.

Симулируется жизнедеятельность пчёл в улье. В начале игры дан улей и несколько пчёл разных видов (собиратели, воины, переработчики мёда). Собиратели облетают поле, собирая нектар с цветов и перенося его в улей. Переработчики конвертируют нектар в мёд, являющийся внутриигровой валютой. Полученный мёд игрок собирает вручную.

Раз в n секунд начинается атака на улей. Всего в игре 5 вражеских волн. Врагами являются осы, нападающие на пчёл и разрушающие улей. На финальной волне на улей нападает шершень, не атакующий пчёл, но имеющий повышенное число очков здоровья. Отбивать вражеские атаки способны пчёлы-воины.

В улье доступен магазин, в котором за собранный мёд игрок может улучшать улей (параметр прочности), покупать новых пчёл и повышать уровень всех пчёл определённого класса.

Игра завершается при победе пчёл над шершнем (победа), либо при разрушении улья или смерти всех пчёл (поражение). Цель игрока – довести пчёл до победного конца игры.

## Условия выполнения и ограничения

Программа должна быть реализована на языке, полностью поддерживающем объектно-ориентированный подход. Иерархия классов должна иметь не менее трёх уровней. Игра должна длится около пяти минут.

# 2 ПРОЕКТНОЕ РЕШЕНИЕ

## 2.1 Абстрагирование и выделение объектов

При анализе программы были выделены следующие объекты:

* пчелы (три типа);
* осы;
* шершень;
* цветы;
* улей;

Для их реализации в системе были выделены следующие классы:

FlyingEntity – обобщающий класс для всех летающих насекомых, задействованных в игре.

Bee – обобщающий класс для всех видов пчёл (собиратели, воины, переработчики).

BeePollinator – пчёлы-собиратели (опылители). Ищут цветы, летят к нм, собирают нектар и передают его в улей для дальнейшей обработки.

BeeWarrior – пчёлы-воины. В основном неактивны, однако во время вражеских волн атакуют врагов (ос, шершня), защищая улей.

BeeRecycler – пчёлы-переработчики. Полученный опылителями нектар конвертируют в мёд, который переносят в бочки, из которых игрок может его собрать.

Enemy – обобщающий класс для вражеских насекомых (осы, шершень).

Wasp – осы, основные враги пчёл. Направляются к улью, чтобы уничтожить его, попутно атакуя пчёл вокруг себя.

Hornet – шершень, финальный босс игры. Не атакует пчёл, однако обладает огромным числом очков здоровья и стремится уничтожить улей.

StaticObject – обобщающий класс для неподвижных (статических) объектов.

Flower – обобщающий класс для цветов. Цветы случайно генерируются на поле, обладают некоторым запасом нектара. После опыления исчезают.

Chamomile – ромашки. Содержат 1-2 очков нектара.

Violet – фиалки. Содержат 1-4 очка нектара.

Chamomile – подсолнухи. Содержат 1-6 очков нектара.

Tree – декоративные деревья, расположенные на поле игры.

Barrel – бочки, в которые пчёлы-переработчики складывают мёд. По нажатию мышью игрок собирает из них мёд.

BeeSpawner – объект для генерации новых пчёл в улье. Срабатывает в начале игры и при покупке новой пчелы.

EnemySpawner – объект для генерации новых врагов. Также отвечает за систему вражеских волн.

FlowerSpawner – объект, генерирующий на поле новые цветы. Следит, чтобы количество цветов не превышало максимума.

Hive – улей. В нём пчёлы-собиратели складируют мёд. При уничтожении улья игра завершается поражением.

BarrelSpawner – объект, генерирующий бочки для новых пчёл-переработчиков.

CameraControl – объект для управления игровой камерой.

Также программе необходим ряд классов для системы покупки пчёл (Buying System), пользовательского интерфейса (UI Panel Controller) и системы прокачки (Upgrader).

## 2.2 Построение иерархии классов

На основе полученной абстракции было построено графическое отображение иерархии классов, представленное на рисунке А.1.

## 2.3 Построение информационной модели

Диаграмма классов представлена на рисунке А.2.

## 2.4 Описание жизненного цикла программы

Диаграмма жизненного цикла программы представлена на рисунке А.3.

## 2.5 Описание жизненного цикла одного из объектов

Диаграмма жизненного цикла объектов представлена на рисунке А.4. В качестве примера были взяты объекты класса BeeWarrior.

## 2.6 Диаграмма переходов состояний

Диаграмма переходов состояний объектов представлена на рисунке А.5. В качестве примера были взяты объекты класса BeeWarrior.

## 2.7 Диаграмма потоков данных и действий

Диаграмма потоков данных и действий представлена на рисунке А.6.

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## 3.1 Обоснование выбора системы программирования

В процессе планирования было принято решение реализовывать проект с помощью игрового движка Unity (версия 2021.3.21f1). Причины выбора были следующими:

* язык программирования C#, имеющий много общего с изучаемым ранее языком C++;
* готовая среда разработки, поддерживающая автоматический рендер изображений;
* простота обучения для неопытных разработчиков;
* большой набор документации и вспомогательного материала от других разработчиков;
* бесплатность;

Для редактирования и компилирования программного кода была выбрана IDE Visual Studio Community 2022. Причины выбора таковы:

* плагин, интегрирующий среду с Unity C#;
* широкий набор инструментов для облегчения написания и редактирования программного кода;
* приятный пользовательский интерфейс;

Разработку было решено вести на ОС Windows.

## 3.2 Описание реализации основных классов и их методов

В ходе разработки были описаны все классы, соответствующие разделу 3.1: Bee, Wasp, Flower, Hornet, Barrel, Hive и др. В процессе проектирования были применены основные принципы ООП.

Все летающие объекты наследуются от класса FlyingEntity. От него расходятся две ветви – пчёлы (Bee) и их враги (Enemy). От класса Bee наследуются классы BeeWarrior, BeePollinator и BeeRecycler. От класса Enemy – классы Wasp и Hornet. При наследовании классам-потомкам передаются поля и методы классов-предков, что является реализацией принципа наследования.

Некоторые методы в процессе наследования переопределяются. Так классы FlyingEntity, Enemy и Bee являются абстрактными, поскольку в них есть чисто виртуальный метод Fly, определённый только у потомков. Таким образом, все потомки класса FlyingEntity реализуют один и тот же метод Fly различными способами. Этот пример иллюстрирует реализацию принципа полиморфизма.

Принцип инкапсуляции был применён через грамотное использование модификаторов доступа. Для методов и полей, недоступных извне, но наследуемых потомками объекта, был установлен модификатор protected (например, очки здоровья и летающих объектов). Вызываемые другими объектами методы были обозначены как public (например, метод TakeDamage и насекомых, вызываемый другими насекомыми при битве). Остальное было обозначено как private (ненаследуемые поля и методы).

## 3.3 Общее описание взаимодействия программных продуктов

Программа состоит из множества компонентов, поддерживаемых игровым движком Unity. Есть две сцены – основная и для главного меню. На основной сцене располагаются игровые объекты, взаимодействующие друг с другом. Некоторые объекты поддерживают интерактивное взаимодействие с игроком, позволяя тому управлять игровым процессом.

У каждого объекта есть набор компонентов, управляющих их состоянием и поведением в системе. Практически каждому объекту соответствуют определённый скрипт – программный код, отвечающий за функциональность объекта.

Также в системе есть скриптовые объекты, отвечающие за хранение и передачу параметров, соответствующих объектам – пчёлам, цветам, осам, улью и др.

Отдельными элементами системы являются ресурсы – используемая в игре музыка и звуки, графические изображения, шрифты и др.

## 3.4 Интерфейс пользователя

Игра начинается в главном меню (Рисунок 1). В нём доступны две кнопки – начать игру (в центре) и выйти из игры (крестик в углу).



Рисунок 1 – Главное меню игры

При начале игры игроку доступно игровое поле (Рисунок 2). В его центре находится улей, также доступна кнопка паузы.



Рисунок 2 – Игровое поле

При нажатии на улей происходит переход в магазин (Рисунок 3). В нём можно приобрести новых пчёл, увеличить уровень пчёл и улья.

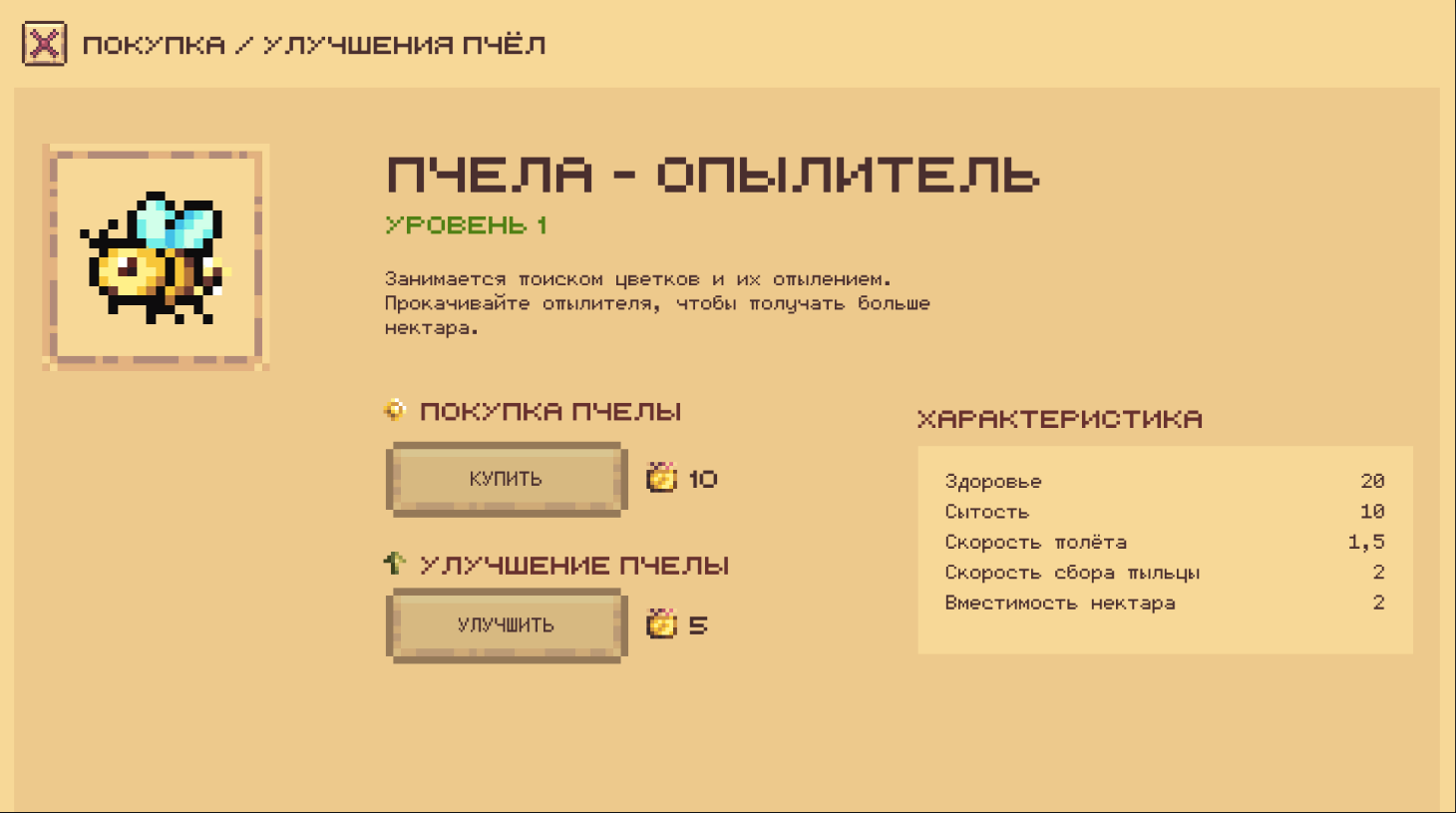


Рисунок 3 – Магазин

При нажатии на кнопку паузы открывается небольшое меню, из которого можно выйти из игры в главное меню, начать новую игру или продолжить текущую (Рисунок 4).



Рисунок 4 – Меню паузы

В случае, если улей был разрушен (целостность опустилась до нуля) или все пчёлы на поле были убиты игра завершается поражением, о котором сигнализирует соответствующий экран (Рисунок 5).

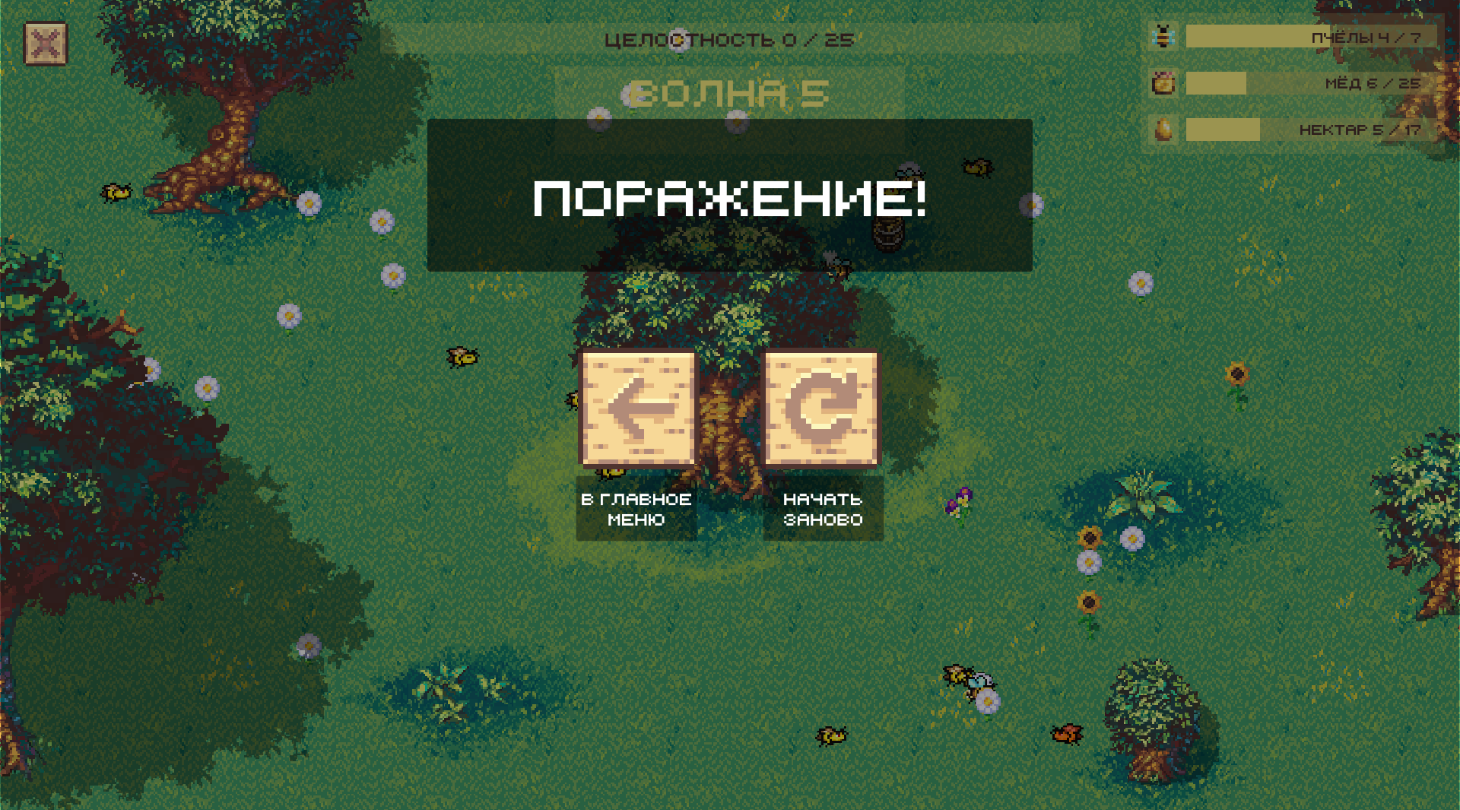


Рисунок 5 – Экран поражения

В случае, если игроку удалось продержаться до конца финальной волны и победи главного врага – шершня, игра завершается победой, о которой сигнализирует соответствующий экран (Рисунок 5).

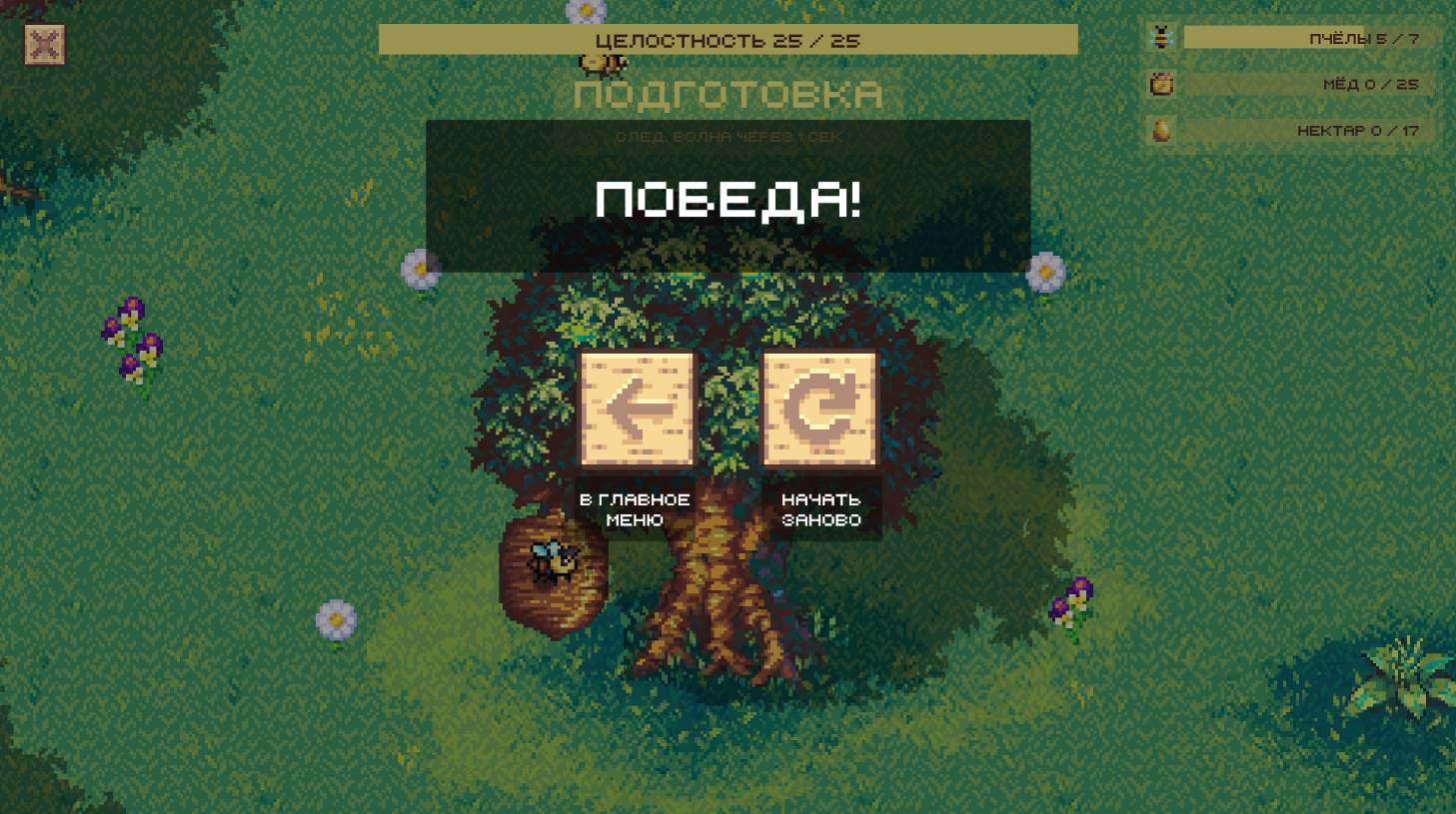


Рисунок 6 – Экран победы

## 3.5 Критерии качества программной системы

К основным качествам программного продукта были отнесены:

* функциональность;
* надёжность;
* эффективная трата ресурсов;
* модифицируемость;
* удобство использования;
* увлекательность;

# ЗАКЛЮЧЕНИЕ

Данный курсовой проект стал итогом изучения курса «Объектно-ориентированное программирование» в рамках образовательной программы института. Одной из целей работы было применить все полученные знания для разработки полноценного программного продукта.

В ходе написания программы были использованы навыки абстрагирования и проектирования системы, применены основные принципы ООП: наследование, инкапсуляция, полиморфизм. Также был получен опыт работы с игровым движком Unity.

В результате была создана игра, код которой написан на языке C# в связке с движком Unity. Игра представляет собой симуляцию жизнедеятельности пчёл в дикой природе с элементами стратегии, где игрок использует полученный мёд для улучшения улья и характеристик пчёл для того, чтобы подготовить их к нападению насекомых-противников.

Игрок может следить за взаимодействием пчёл с другими игровыми объектами (улей, осы, цветы, бочки, шершень), смотреть их характеристики, собирать мёд и использовать его в магазине для улучшения старых или создания новых пчёл.

Полученные навыки могут быть применены для дальнейшей реализации задач, которые подразумевают применение при разработке объектно-ориентированного подхода.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Почувствуй класс / Мейер Б.; пер. с англ. под ред. В.А. Биллига. — М.: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011. —775 с.: ил., табл.
2. Геймдизайн: Как создать игру, в которую будут играть все / Джесси Ш. - Альпина Паблишер. 2008 – 435 с.
3. Мюллер, Дж. П. C# для чайников / Дж. П. Мюллер. - СПб.: Вильямс, 2016. - 432 с.
4. Ширигатти, Джексон Луис. Разработка 2D игр для начинающих: Подход с помощью платформы Unity и языка программирования C#. Москва: ДМК Пресс, 2020. ISBN 978-5-97060-953-1.
5. Официальный справочник по разработке на игровом движке Unity [Электронный ресурс] — <https://unity.com/ru/learn>.
6. Методические указания к лабораторным работам по дисциплине “Объектно-ориентированное программирование” для студентов направления 09.03.02 — “Информационные системы и технологии”/ Сост. Т.И. Сметанина, А.К. Забаштанский — Севастополь: Изд-во СевГУ 2017. — 80с.
7. Cистема вопросов и ответов о программировании StackOverFlow (на русском) [Электронный ресурс] — <https://ru.stackoverflow.com/>
8. Вопросы и ответы по IT-темам: Overcoder [Электронный ресурс] — <https://overcoder.net>
9. Система вопросов и ответов о программировании CodeNet [Электронный ресурс] — <http://forum.codenet.ru/>
10. Форум программистов и сисадминов: Киберфорум [Электронный ресурс] — <https://www.cyberforum.ru/cpp-beginners/>

# ПРИЛОЖЕНИЕ А

UML-ДИАГРАММЫ

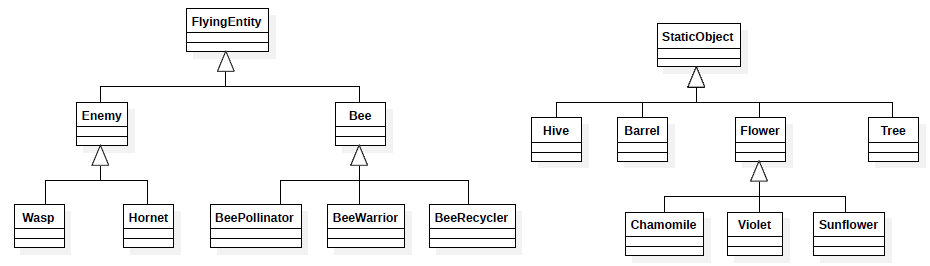


Рисунок А.1 – Иерархия классов

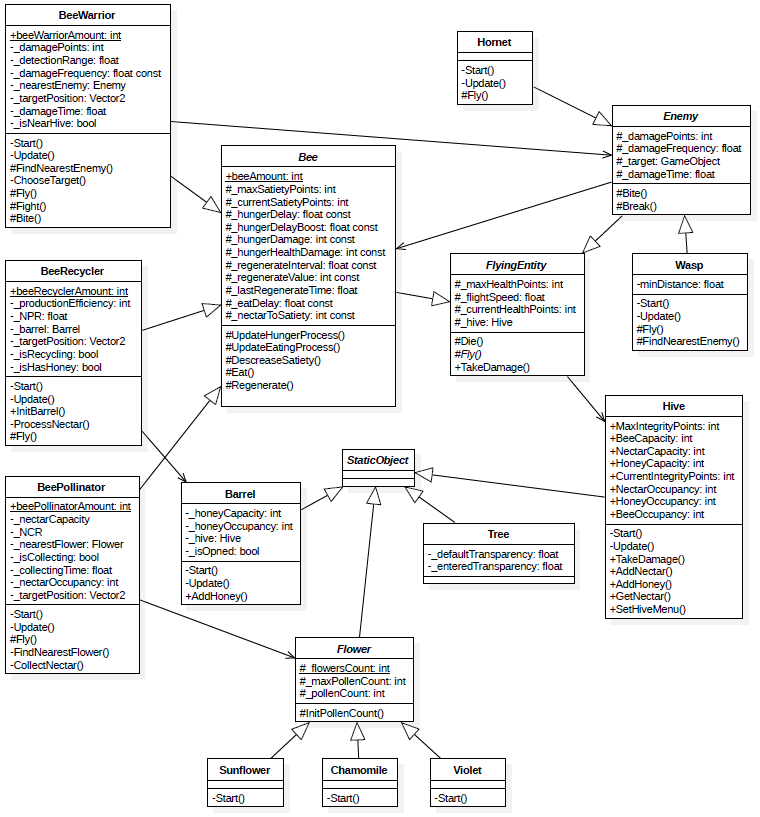


Рисунок А.2 – Диаграмма классов

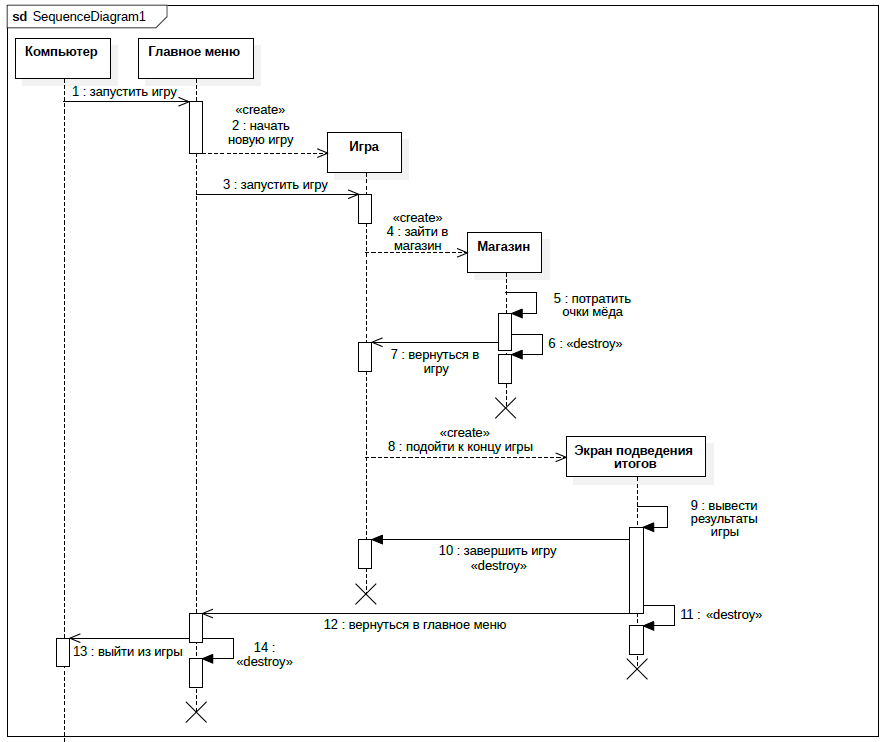


Рисунок А.3 – Жизненный цикл программы

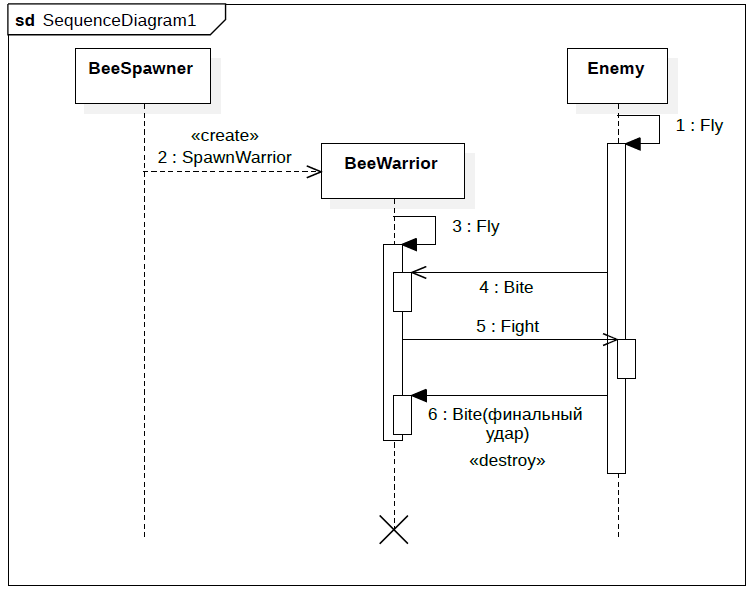


Рисунок А.4 – Жизненный цикл объектов класса BeeWarrior

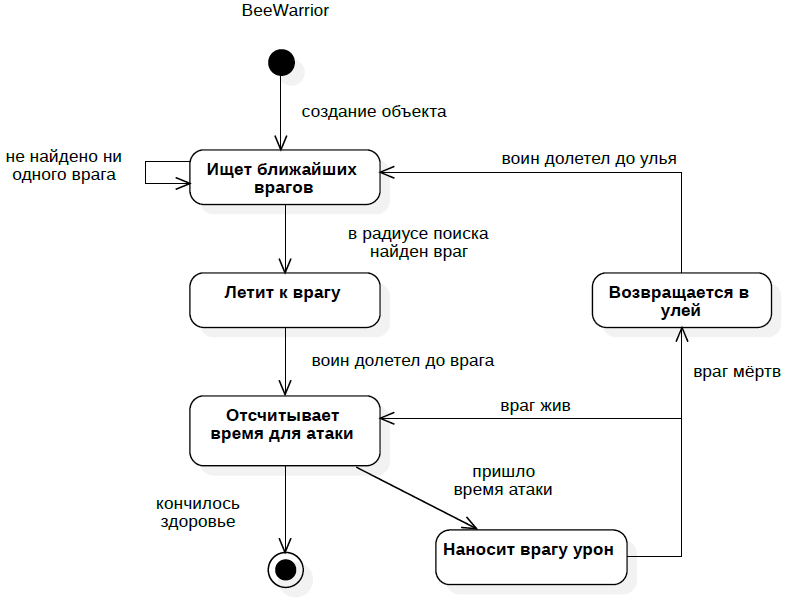


Рисунок А.5 – Диаграмма переходов состояний объектов класса BeeWarrior

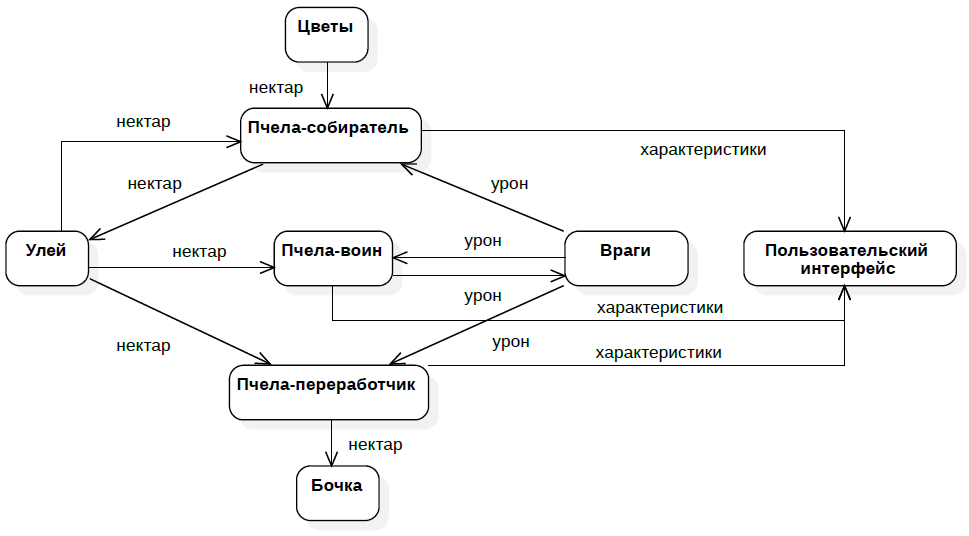


Рисунок А.6 – Диаграмма потоков данных и действий

# ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД

FlyingEntity.cs

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

public abstract class FlyingEntity : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler

{

[SerializeField] protected GameObject \_hintPanel;

[SerializeField] protected Text \_nameText;

[SerializeField] private Slider \_healthPointsSlider;

protected int \_maxHealthPoints;

protected float \_flightSpeed;

protected string \_name;

protected int \_currentHealthPoints;

protected abstract void Fly();

protected virtual void Die()

{

Destroy(gameObject);

}

public virtual void TakeDamage(int damage)

{

\_currentHealthPoints -= damage;

\_currentHealthPoints = Mathf.Clamp(\_currentHealthPoints, 0, \_maxHealthPoints);

UpdateHealthPointsSlider();

if (\_currentHealthPoints <= 0)

{

Die();

}

}

protected virtual void SetHintPanelSettings()

{

\_nameText.text = \_name;

\_hintPanel.SetActive(false);

UpdateHealthPointsSlider();

}

protected void UpdateHealthPointsSlider()

{

\_healthPointsSlider.value = (float)\_currentHealthPoints / \_maxHealthPoints;

}

public void OnPointerEnter(PointerEventData eventData)

{

\_hintPanel.SetActive(true);

}

public void OnPointerExit(PointerEventData eventData)

{

\_hintPanel.SetActive(false);

}

}

Enemy.cs

using UnityEngine;

public abstract class Enemy : FlyingEntity

{

protected int \_damagePoints;

protected float \_damageFrequency;

protected GameObject \_target;

protected Hive \_hive;

protected float \_damageTime = 0f;

protected void Bite()

{

//\_takeDamageAnim.SetBool("TakeDamageWasp", true);

if (Time.time - \_damageTime > \_damageFrequency)

{

\_target.GetComponent<Bee>().TakeDamage(\_damagePoints);

\_damageTime = Time.time;

}

//\_takeDamageAnim.SetBool("TakeDamageWasp", false);

}

protected void Break()

{

if (Time.time - \_damageTime > \_damageFrequency)

{

\_hive.TakeDamage(\_damagePoints);

\_damageTime = Time.time;

}

}

}

Wasp.cs

using Unity.VisualScripting;

using UnityEngine;

public class Wasp : Enemy

{

[SerializeField] protected EnemyData \_waspData;

new private static int \_maxHealthPoints;

new private static float \_flightSpeed;

new private static int \_damagePoints;

new private static float \_damageFrequency;

private SpriteRenderer \_spriteRenderer;

private Animator \_takeDamageAnim;

private float minDistance = 0f;

public static int MaxHealthPoints

{

get { return \_maxHealthPoints; }

set { \_maxHealthPoints = value; }

}

public static float FlightSpeed

{

get { return \_flightSpeed; }

set { \_flightSpeed = value; }

}

public static int damagePoints

{

get { return \_damagePoints; }

set { \_damagePoints = value; }

}

public static float damageFrequency

{

get { return \_damageFrequency; }

set { \_damageFrequency = value; }

}

private void Start()

{

LoadData(\_waspData);

\_takeDamageAnim = GetComponent<Animator>();

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hive = FindObjectOfType<Hive>();

\_currentHealthPoints = \_maxHealthPoints;

SetHintPanelSettings();

}

private void Update()

{

FindNearestTarget();

Fly();

}

protected override void Fly()

{

if (minDistance > 0.5)

{

transform.position = Vector2.MoveTowards(transform.position, \_target.transform.position, \_flightSpeed \* Time.deltaTime);

\_spriteRenderer.flipX = \_target.transform.position.x >= transform.position.x;

}

else if (\_target.CompareTag("bee"))

{

Bite();

}

else if (\_target.CompareTag("hive"))

{

Break();

}

}

protected void FindNearestTarget()

{

Bee[] bees = FindObjectsOfType<Bee>();

minDistance = Mathf.Infinity;

foreach (Bee bee in bees)

{

float distance = Vector2.Distance(transform.position, bee.transform.position);

if (distance < minDistance)

{

minDistance = distance;

\_target = bee.gameObject;

}

}

float hiveDistance = Vector2.Distance(transform.position, \_hive.transform.position);

if (hiveDistance < minDistance)

{

minDistance = hiveDistance;

\_target = \_hive.gameObject;

}

}

private void LoadData(EnemyData data)

{

\_name = data.name;

\_maxHealthPoints = Mathf.Max(\_maxHealthPoints, data.healthPoints);

base.\_maxHealthPoints = \_maxHealthPoints;

\_flightSpeed = Mathf.Max(\_flightSpeed, data.flightSpeed);

base.\_flightSpeed = \_flightSpeed;

\_damagePoints = Mathf.Max(\_damagePoints, data.damagePoints);

base.\_damagePoints = \_damagePoints;

\_damageFrequency = Mathf.Max(\_damageFrequency, data.damageFrequency);

base.\_damageFrequency = \_damageFrequency;

}

}

Hornet.cs

using Unity.VisualScripting;

using UnityEngine;

public class Hornet : Enemy

{

[SerializeField] protected EnemyData \_enemyData;

private float hiveDistance = 0f;

new private static int \_maxHealthPoints;

new private static float \_flightSpeed;

new private static int \_damagePoints;

new private static float \_damageFrequency;

public static int MaxHealthPoints

{

get { return \_maxHealthPoints; }

set { \_maxHealthPoints = value; }

}

public static float FlightSpeed

{

get { return \_flightSpeed; }

set { \_flightSpeed = value; }

}

public static int damagePoints

{

get { return \_damagePoints; }

set { \_damagePoints = value; }

}

public static float damageFrequency

{

get { return \_damageFrequency; }

set { \_damageFrequency = value; }

}

private SpriteRenderer \_spriteRenderer;

private void Start()

{

LoadData(\_enemyData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hive = FindObjectOfType<Hive>();

\_currentHealthPoints = \_maxHealthPoints;

SetHintPanelSettings();

}

private void Update()

{

hiveDistance = Vector2.Distance(transform.position, \_hive.transform.position);

if (hiveDistance > 0.1)

{

Fly();

}

else

{

Break();

}

}

protected override void Fly()

{

transform.position = Vector2.MoveTowards(transform.position, \_hive.transform.position, \_flightSpeed \* Time.deltaTime);

\_spriteRenderer.flipX = \_hive.transform.position.x >= transform.position.x;

}

private void LoadData(EnemyData data)

{

\_name = data.name;

\_maxHealthPoints = Mathf.Max(\_maxHealthPoints, data.healthPoints);

base.\_maxHealthPoints = \_maxHealthPoints;

\_flightSpeed = Mathf.Max(\_flightSpeed, data.flightSpeed);

base.\_flightSpeed = \_flightSpeed;

\_damagePoints = Mathf.Max(\_damagePoints, data.damagePoints);

base.\_damagePoints = \_damagePoints;

\_damageFrequency = Mathf.Max(\_damageFrequency, data.damageFrequency);

base.\_damageFrequency = \_damageFrequency;

}

}

Bee.cs

using UnityEngine;

using UnityEngine.UI;

public abstract class Bee : FlyingEntity

{

public static int beeAmount = 0;

[SerializeField] private Slider \_satietyPointsSlider;

protected int \_maxSatietyPoints;

protected int \_currentSatietyPoints;

protected const float \_hungerDelay = 7;

protected const float \_hungerDelayBoost = 3;

protected const int \_hungerDamage = 1;

protected const int \_hungerHealthDamage = 1;

protected const float \_regenerateInterval = 1;

protected const int \_regenerateValue = 1;

protected float \_lastRegenerateTime;

protected const float \_eatDelay = 8;

protected const int \_nectarToSatiety = 4;

protected Hive \_hive;

protected void UpdateHungerProcess(float delay)

{

CancelInvoke("DecreaseSatiety");

InvokeRepeating("DecreaseSatiety", 0.5f, delay);

}

protected void UpdateEatingProcess(float delay)

{

CancelInvoke("Eat");

InvokeRepeating("Eat", 0.5f, delay);

}

protected virtual void DecreaseSatiety()

{

if (\_currentSatietyPoints > 0)

{

\_currentSatietyPoints -= \_hungerDamage;

UpdateSatietyPointsSlider();

}

else

{

TakeDamage(\_hungerHealthDamage);

}

}

protected virtual void Eat()

{

if (\_currentSatietyPoints < \_maxSatietyPoints)

{

int needSatiety = \_maxSatietyPoints - \_currentSatietyPoints;

int needNectar = needSatiety / \_nectarToSatiety;

if (needNectar > 0)

{

\_currentSatietyPoints += \_hive.GetNectar(needNectar) \* \_nectarToSatiety;

}

UpdateSatietyPointsSlider();

}

}

protected virtual void Regenerate()

{

if (\_currentHealthPoints < \_maxHealthPoints && \_currentSatietyPoints > 0)

{

float percent = (float)\_currentSatietyPoints / \_maxSatietyPoints;

float interval = \_regenerateInterval \* (2 - percent);

if (Time.time - \_lastRegenerateTime > interval)

{

\_currentHealthPoints += \_regenerateValue;

UpdateHealthPointsSlider();

if (\_currentHealthPoints > \_maxHealthPoints)

{

\_currentHealthPoints = \_maxHealthPoints;

}

\_lastRegenerateTime = Time.time;

}

}

}

protected void UpdateSatietyPointsSlider()

{

\_satietyPointsSlider.value = (float)\_currentSatietyPoints / \_maxSatietyPoints;

}

protected override void SetHintPanelSettings()

{

base.SetHintPanelSettings();

UpdateSatietyPointsSlider();

}

protected override void Die()

{

base.Die();

beeAmount -= 1;

}

}

BeePollinator.cs:

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class BeePollinator : Bee

{

public static int beePollinatorAmount;

[SerializeField] private BeePollinatorData \_beePollinatorData;

private SpriteRenderer \_spriteRenderer;

private Flower \_nearestFlower;

new private static int \_maxHealthPoints = 0;

new private static int \_maxSatietyPoints = 0;

new private static float \_flightSpeed = 0;

private static int \_nectarCapacity = 0;

private static float \_NCR = 0; // nectar collection rate

private Animator anim;

public static int MaxHealthPoints

{

get { return \_maxHealthPoints; }

set { \_maxHealthPoints = value; }

}

public static int MaxSatietyPoints

{

get { return \_maxSatietyPoints; }

set { \_maxSatietyPoints = value; }

}

public static float FlightSpeed

{

get { return \_flightSpeed; }

set { \_flightSpeed = value; }

}

public static int NectarCapacity

{

get { return \_nectarCapacity; }

set { \_nectarCapacity = value; }

}

public static float NCR

{

get { return \_NCR; }

set { \_NCR = value; }

}

private bool \_isCollecting;

private float \_collectingTime = 0f;

private int \_nectarOccupancy = 0;

private Vector2 \_targetPosition;

private void Start()

{

LoadData(\_beePollinatorData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hive = FindObjectOfType<Hive>();

\_isCollecting = false;

\_currentHealthPoints = \_maxHealthPoints;

\_currentSatietyPoints = \_maxSatietyPoints;

SetHintPanelSettings();

UpdateHungerProcess(\_hungerDelay);

UpdateEatingProcess(\_eatDelay);

anim = GetComponent<Animator>();

}

private void Update()

{

if (\_nearestFlower == null)

{

FindNearestFlower();

}

if (\_nearestFlower != null)

{

\_nearestFlower.gameObject.tag = "flower\_busy";

float nearestDistance = Vector2.Distance(transform.position, \_nearestFlower.transform.position);

if (nearestDistance < 0.1f)

{

CollectNectar(\_nearestFlower);

}

}

if (\_nectarOccupancy > 0)

{

anim.SetBool("isPollen", true);

}

else

{

anim.SetBool("isPollen", false);

}

Fly();

Regenerate();

SpriteRender();

}

protected override void Fly()

{

if (\_nectarOccupancy < \_nectarCapacity && \_nearestFlower != null && !\_isCollecting)

{

\_targetPosition = \_nearestFlower.transform.position;

}

if (\_nearestFlower == null)

{

if (\_nectarOccupancy > 0)

{

\_targetPosition = \_hive.transform.position;

}

else

{

\_targetPosition = transform.position;

}

}

transform.position = Vector2.MoveTowards(transform.position, \_targetPosition, \_flightSpeed \* Time.deltaTime);

}

private void FindNearestFlower()

{

Flower[] flowers = FindObjectsOfType<Flower>();

float minDistance = Mathf.Infinity;

foreach (Flower flower in flowers)

{

int flowerPollenAmount = flower.GetComponent<Flower>().PollenCount;

if (flowerPollenAmount != 0)

{

float distance = Vector2.Distance(transform.position, flower.transform.position);

int futureNectar = \_nectarOccupancy + flowerPollenAmount;

if (flower.gameObject.tag == "flower" && futureNectar <= \_nectarCapacity && distance < minDistance)

{

minDistance = distance;

\_nearestFlower = flower;

}

}

}

}

private void CollectNectar(Flower \_flower)

{

\_isCollecting = true;

if (\_collectingTime < \_NCR)

{

\_collectingTime += Time.deltaTime;

}

else

{

\_nectarOccupancy += \_flower.GetComponent<Flower>().PollenCount;

Destroy(\_flower.gameObject);

Flower.FlowersCount -= 1;

\_flower = null;

\_collectingTime = 0;

\_isCollecting = false;

}

}

private void SpriteRender()

{

\_spriteRenderer.flipX = \_targetPosition.x >= transform.position.x;

}

private void LoadData(BeePollinatorData data)

{

\_name = data.name;

\_maxHealthPoints = Mathf.Max(\_maxHealthPoints, data.healthPoints);

base.\_maxHealthPoints = \_maxHealthPoints;

\_maxSatietyPoints = Mathf.Max(\_maxSatietyPoints, data.satietyPoints);

base.\_maxSatietyPoints = \_maxSatietyPoints;

\_flightSpeed = Mathf.Max(\_flightSpeed, data.flightSpeed);

base.\_flightSpeed = \_flightSpeed;

\_NCR = Mathf.Max(\_NCR, data.NCR);

\_nectarCapacity = Mathf.Max(\_nectarCapacity, data.nectarCapacity);

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.tag == "hive" && \_nectarOccupancy > 0)

{

\_hive.AddNectar(\_nectarOccupancy);

\_nectarOccupancy = 0;

}

}

}

BeeWarrior.cs

using UnityEngine;

using static UnityEngine.GraphicsBuffer;

public class BeeWarrior : Bee

{

public static int beeWarriorAmount;

[SerializeField] private BeeWarriorData \_beeWarriorData;

new private static int \_maxHealthPoints = 0;

new private static int \_maxSatietyPoints = 0;

new private static float \_flightSpeed = 0;

private static int \_damagePoints = 0;

private static float \_detectionRange = 0;

public static int MaxHealthPoints

{

get { return \_maxHealthPoints; }

set { \_maxHealthPoints = value; }

}

public static int MaxSatietyPoints

{

get { return \_maxSatietyPoints; }

set { \_maxSatietyPoints = value; }

}

public static float FlightSpeed

{

get { return \_flightSpeed; }

set { \_flightSpeed = value; }

}

public static int DamagePoints

{

get { return \_damagePoints; }

set { \_damagePoints = value; }

}

public static float DetectionRange

{

get { return \_detectionRange; }

set { \_detectionRange = value; }

}

private const float \_damageFrequency = 1.0f;

private Enemy \_nearestEnemy;

private SpriteRenderer \_spriteRenderer;

private Vector2 \_targetPosition;

[SerializeField] // временно

private float \_damageTime;

private bool \_isNearHive;

void Start()

{

LoadData(\_beeWarriorData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hive = FindAnyObjectByType<Hive>();

\_isNearHive = false;

\_currentHealthPoints = \_maxHealthPoints;

\_currentSatietyPoints = \_maxSatietyPoints;

SetHintPanelSettings();

UpdateHungerProcess(\_hungerDelay);

UpdateEatingProcess(\_eatDelay);

}

void Update()

{

if (\_nearestEnemy == null)

{

FindNearestEnemy();

}

ChooseTarget();

Fly();

Regenerate();

}

private void FindNearestEnemy()

{

Enemy[] enemies = FindObjectsOfType<Enemy>();

float closestDistance = Mathf.Infinity;

foreach (Enemy enemy in enemies)

{

float distance = Vector2.Distance(transform.position, enemy.transform.position);

if (distance < closestDistance && distance <= \_detectionRange)

{

closestDistance = distance;

\_nearestEnemy = enemy;

}

}

}

protected void ChooseTarget()

{

if (\_nearestEnemy != null)

{

\_targetPosition = \_nearestEnemy.transform.position;

}

else if (!\_isNearHive)

{

\_targetPosition = \_hive.transform.position;

}

}

protected override void Fly()

{

float distance = Vector2.Distance(transform.position, \_targetPosition);

if (distance > 0.4 || \_nearestEnemy == null)

{

transform.position = Vector2.MoveTowards(transform.position, \_targetPosition, \_flightSpeed \* Time.deltaTime);

\_spriteRenderer.flipX = transform.position.x < \_targetPosition.x;

}

else

{

Fight();

}

}

protected void Fight()

{

if (Time.time - \_damageTime > \_damageFrequency)

{

Bite(\_nearestEnemy);

\_damageTime = Time.time;

}

}

private void Bite(Enemy enemy)

{

enemy.TakeDamage(\_damagePoints);

}

public void LoadData(BeeWarriorData data)

{

\_name = data.name;

\_maxHealthPoints = Mathf.Max(\_maxHealthPoints, data.healthPoints);

base.\_maxHealthPoints = \_maxHealthPoints;

\_maxSatietyPoints = Mathf.Max(\_maxSatietyPoints, data.satietyPoints);

base.\_maxSatietyPoints = \_maxSatietyPoints;

\_flightSpeed = Mathf.Max(\_flightSpeed, data.flightSpeed);

base.\_flightSpeed = \_flightSpeed;

\_damagePoints = Mathf.Max(\_damagePoints, data.damagePoints);

\_detectionRange = Mathf.Max(\_detectionRange, data.detectionRange);

}

private void OnTriggerExit2D(Collider2D collision)

{

if (collision.CompareTag("hive"))

{

//\_spriteRenderer.enabled = true;

\_isNearHive = false;

UpdateHungerProcess(\_hungerDelay); // faster speed hunger process

}

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("hive"))

{

//\_spriteRenderer.enabled = false;

\_isNearHive = true;

\_targetPosition = transform.position;

UpdateHungerProcess(\_hungerDelay + \_hungerDelayBoost); // slower speed hunger process

}

}

}

BeeRecycler.cs

using UnityEngine;

using static UnityEngine.GraphicsBuffer;

public class BeeRecycler : Bee

{

public static int beeRecyclerAmount;

[SerializeField] private BeeRecyclerData \_beeRecyclerData;

new private static int \_maxHealthPoints = 0;

new private static int \_maxSatietyPoints = 0;

new private static float \_flightSpeed = 0;

private static int \_productionEfficiency = 0;

private static float \_NPR = 0;

public static int MaxHealthPoints

{

get { return \_maxHealthPoints; }

set { \_maxHealthPoints = value; }

}

public static int MaxSatietyPoints

{

get { return \_maxSatietyPoints; }

set { \_maxSatietyPoints = value; }

}

public static float FlightSpeed

{

get { return \_flightSpeed; }

set { \_flightSpeed = value; }

}

public static int ProductionEfficiency

{

get { return \_productionEfficiency; }

set { \_productionEfficiency = value; }

}

public static float NPR

{

get { return \_NPR; }

set { \_NPR = value; }

}

private Animator anim;

private Barrel \_barrel;

private SpriteRenderer \_spriteRenderer;

private Vector2 \_targetPosition;

private bool \_isRecycling;

private bool \_isHasHoney;

private void Start()

{

LoadData(\_beeRecyclerData);

\_hive = FindObjectOfType<Hive>();

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_isRecycling = false;

\_isHasHoney = false;

\_spriteRenderer.enabled = false;

\_currentHealthPoints = \_maxHealthPoints;

\_currentSatietyPoints = \_maxSatietyPoints;

\_flightSpeed = (Vector2.Distance(\_hive.transform.position, \_barrel.transform.position) / \_NPR) \* 2;

SetHintPanelSettings();

UpdateHungerProcess(\_hungerDelay);

UpdateEatingProcess(\_eatDelay);

anim = GetComponent<Animator>();

}

private void Update()

{

anim.SetBool("isHoney", \_isHasHoney);

ProcessNectar();

Fly();

Regenerate();

SpriteRender();

}

public void InitBarrel(Barrel barrel)

{

\_barrel = barrel;

}

private void ProcessNectar()

{

if (!\_isRecycling && Hive.NectarOccupancy > 0)

{

\_hive.GetNectar(1);

\_isRecycling = true;

\_isHasHoney = true;

\_spriteRenderer.enabled = true;

\_targetPosition = \_barrel.transform.position;

}

if (\_isRecycling)

{

if (\_isHasHoney)

{

float barrelDistance = Vector2.Distance(transform.position, \_barrel.transform.position);

if (barrelDistance < 0.1f)

{

\_barrel.AddHoney(1 \* \_productionEfficiency);

\_isHasHoney = false;

\_targetPosition = \_hive.transform.position;

}

}

else

{

float hiveDistance = Vector2.Distance(transform.position, \_hive.transform.position);

if (hiveDistance < 0.1f)

{

\_isRecycling = false;

\_spriteRenderer.enabled = false;

}

}

}

}

protected override void Fly()

{

if (\_isRecycling)

{

transform.position = Vector2.MoveTowards(transform.position, \_targetPosition, \_flightSpeed \* Time.deltaTime);

}

}

protected override void Die()

{

base.Die();

Destroy(\_barrel.gameObject);

}

private void SpriteRender()

{

\_spriteRenderer.flipX = \_targetPosition.x >= transform.position.x;

}

public void LoadData(BeeRecyclerData data)

{

\_name = data.name;

\_maxHealthPoints = Mathf.Max(\_maxHealthPoints, data.healthPoints);

base.\_maxHealthPoints = \_maxHealthPoints;

\_maxSatietyPoints = Mathf.Max(\_maxSatietyPoints, data.satietyPoints);

base.\_maxSatietyPoints = \_maxSatietyPoints;

\_flightSpeed = Mathf.Max(\_flightSpeed, data.flightSpeed);

base.\_flightSpeed = \_flightSpeed;

\_NPR = Mathf.Max(\_NPR, data.NPR);

\_productionEfficiency = Mathf.Max(\_productionEfficiency, data.productionEfficiency);

}

}

StaticObject.cs

using System;

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

public abstract class StaticObject : MonoBehaviour, IPointerClickHandler, IPointerEnterHandler, IPointerExitHandler

{

[SerializeField] protected GameObject \_hintPanel;

[SerializeField] protected Text \_nameText;

[SerializeField] protected GameObject \_borderHint;

[SerializeField] protected Sprite \_defaultSprite;

protected string \_name;

public abstract void OnPointerClick(PointerEventData eventData);

public abstract void OnPointerEnter(PointerEventData eventData);

public abstract void OnPointerExit(PointerEventData eventData);

}

Hive.cs

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

public class Hive : StaticObject

{

[SerializeField] private HiveData \_hiveData;

[SerializeField] private GameObject \_hiveOptionsUIPanel;

[SerializeField] private ExitPanelController \_exitPanel;

[SerializeField] private Slider \_integrityPointsSlider;

private SpriteRenderer \_spriteRenderer;

public static int MaxIntegrityPoints;

public static int BeeCapacity;

public static int NectarCapacity;

public static int HoneyCapacity;

public static int CurrentIntegrityPoints;

public static int NectarOccupancy;

public static int HoneyOccupancy;

public static int BeeOccupancy;

private Animator \_takeDamage;

private void Start()

{

LoadData(\_hiveData);

\_takeDamage = GetComponent<Animator>();

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hintPanel.SetActive(false);

CurrentIntegrityPoints = MaxIntegrityPoints;

}

private void Update()

{

BeeOccupancy = Bee.beeAmount;

}

public virtual void TakeDamage(int damage)

{

\_takeDamage.SetBool("TakeDamage", true);

CurrentIntegrityPoints -= damage;

CurrentIntegrityPoints = Mathf.Clamp(CurrentIntegrityPoints, 0, MaxIntegrityPoints);

UpdateHealthPointsSlider();

if (CurrentIntegrityPoints <= 0)

{

\_exitPanel.ShowResultPanel("ПОРАЖЕНИЕ!");

}

\_takeDamage.SetBool("TakeDamage", false);

}

protected virtual void SetHintPanelSettings()

{

\_nameText.text = \_name;

\_hintPanel.SetActive(false);

UpdateHealthPointsSlider();

}

protected void UpdateHealthPointsSlider()

{

\_integrityPointsSlider.value = (float)CurrentIntegrityPoints / MaxIntegrityPoints;

}

public void AddNectar(int amount)

{

NectarOccupancy += amount;

if (NectarOccupancy > NectarCapacity)

{

NectarOccupancy = NectarCapacity;

}

}

public void AddHoney(int amount)

{

HoneyOccupancy += amount;

if (HoneyOccupancy > HoneyCapacity)

{

HoneyOccupancy = HoneyCapacity;

}

}

public int GetNectar(int amount)

{

if(amount > NectarOccupancy)

{

amount = NectarOccupancy;

}

NectarOccupancy -= amount;

return amount;

}

public override void OnPointerClick(PointerEventData eventData)

{

SetHiveMenu();

\_borderHint.SetActive(true);

\_hintPanel.SetActive(true);

}

public override void OnPointerEnter(PointerEventData eventData)

{

\_borderHint.SetActive(true);

\_hintPanel.SetActive(true);

}

public override void OnPointerExit(PointerEventData eventData)

{

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

public void SetHiveMenu()

{

TimePause();

\_hiveOptionsUIPanel.SetActive(true);

}

public void TimePause()

{

Time.timeScale = 0;

}

public void ContinueTime()

{

Time.timeScale = 1;

}

private void LoadData(HiveData data)

{

\_name = data.name;

MaxIntegrityPoints = data.maxIntegrityPoints;

BeeCapacity = data.beeCapacity;

HoneyCapacity = data.honeyCapacity;

NectarCapacity = data.nectarCapacity;

}

}

HiveRestoreSystem.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class HiveRestoreSystem : MonoBehaviour

{

[SerializeField] private HiveUIPanelController \_hiveUIPanel;

[SerializeField] private int \_price;

[SerializeField] private float \_inflationaryGrowth;

void Start()

{

\_hiveUIPanel.UpdatePriceText(\_price);

}

void Update()

{

if (Hive.CurrentIntegrityPoints < Hive.MaxIntegrityPoints)

{

if (Hive.HoneyOccupancy >= \_price)

{

\_hiveUIPanel.EnableRestoreButton();

}

else

{

\_hiveUIPanel.DisableRestoreButton();

}

}

else

{

\_hiveUIPanel.SetVoidRestore();

}

}

public void Restore()

{

Hive.CurrentIntegrityPoints += 1;

Hive.HoneyOccupancy -= \_price;

UpdatePrice();

\_hiveUIPanel.UpdatePriceText(\_price);

}

private void UpdatePrice()

{

\_price = (int)Mathf.Ceil(\_price\*\_inflationaryGrowth);

}

}

Tree.cs

using UnityEngine;

using UnityEngine.EventSystems;

public class Tree : StaticObject

{

private SpriteRenderer \_spriteRenderer;

private float \_defaultTransparency = 1;

private float \_enteredTransparency = 0.5f;

public override void OnPointerClick(PointerEventData eventData)

{

\_spriteRenderer.color = new Color(255, 255, 255, \_defaultTransparency);

}

public override void OnPointerEnter(PointerEventData eventData)

{

\_spriteRenderer.color = new Color(255, 255, 255, \_enteredTransparency);

}

public override void OnPointerExit(PointerEventData eventData)

{

\_spriteRenderer.color = new Color(255, 255, 255, \_defaultTransparency);

}

// Start is called before the first frame update

void Start()

{

\_spriteRenderer = GetComponent<SpriteRenderer>();

}

// Update is called once per frame

void Update()

{

}

}

MenuController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MenuController : MonoBehaviour

{

public void ChangeScene(string sceneName)

{

LoadSceneManager.instance.LoadScene(sceneName);

}

public void QuitGame()

{

Application.Quit();

}

}

LoadSceneManager.cs:

using System.Collections;

using System.Collections.Generic;

using System.Threading.Tasks;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class LoadSceneManager : MonoBehaviour

{

public static LoadSceneManager instance;

[SerializeField] private GameObject \_loaderCanvas;

[SerializeField] private Slider \_progressSlider;

private float \_targetProgress;

void Awake()

{

if (instance == null)

{

instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

public async void LoadScene(string sceneName)

{

\_progressSlider.value = 0;

\_targetProgress = 0;

var scene = SceneManager.LoadSceneAsync(sceneName);

scene.allowSceneActivation = false;

\_loaderCanvas.SetActive(true);

do

{

await Task.Delay(200);

\_targetProgress = scene.progress;

} while (scene.progress < 0.9f);

scene.allowSceneActivation = true;

\_loaderCanvas.SetActive(false);

}

void Update()

{

\_progressSlider.value = Mathf.MoveTowards(\_progressSlider.value, \_targetProgress, 3 \* Time.deltaTime);

}

}

CameraControl.cs

using UnityEngine;

using System.Collections;

using Unity.VisualScripting;

using UnityEngine.UIElements;

public class CameraControl : MonoBehaviour

{

[SerializeField] private float \_minSize = 2.0f;

[SerializeField] private float \_maxSize = 7.7f;

[SerializeField] private float \_defaultSize = 5.0f;

[SerializeField] private float \_zoomSpeed = 1.0f;

[SerializeField] private float \_moveSpeed = 1.0f;

//[SerializeField] private float \_smoothTime = 0.5f;

[SerializeField] SpriteRenderer \_mapRenderer;

private Camera \_cam;

private Vector3 \_dragOrigin;

private float \_mapMinX, \_mapMaxX, \_mapMinY, \_mapMaxY;

//float \_velocity = 1;

private void Start()

{

\_cam = GetComponent<Camera>();

\_cam.orthographicSize = \_defaultSize;

}

private void Awake()

{

\_mapMinX = \_mapRenderer.transform.position.x - \_mapRenderer.bounds.size.x / 2f;

\_mapMaxX = \_mapRenderer.transform.position.x + \_mapRenderer.bounds.size.x / 2f;

\_mapMinY = \_mapRenderer.transform.position.y - \_mapRenderer.bounds.size.y / 2f;

\_mapMaxY = \_mapRenderer.transform.position.y + \_mapRenderer.bounds.size.y / 2f;

}

private void Update()

{

MoveCamera();

ZoomCamera();

\_cam.transform.position = ClampCamera(\_cam.transform.position);

}

private void MoveCamera()

{

if (Input.GetMouseButtonDown(0))

{

\_dragOrigin = \_cam.ScreenToWorldPoint(Input.mousePosition);

}

if (Input.GetMouseButton(0))

{

Vector3 difference = \_dragOrigin - \_cam.ScreenToWorldPoint(Input.mousePosition);

\_cam.transform.position += difference \* Time.deltaTime \* \_moveSpeed;

}

}

private void ZoomCamera()

{

float scrollValue = Input.GetAxis("Mouse ScrollWheel");

if (scrollValue != 0)

{

float currentSize = \_cam.orthographicSize;

currentSize -= scrollValue \* \_zoomSpeed;

currentSize = Mathf.Clamp(currentSize, \_minSize, \_maxSize);

\_cam.orthographicSize = currentSize;

//\_cam.orthographicSize = Mathf.SmoothDamp(\_cam.orthographicSize, currentSize, ref \_velocity, \_smoothTime);

float cursorDistance = Vector3.Distance(\_cam.transform.position, \_cam.ScreenToWorldPoint(Input.mousePosition));

\_cam.transform.position = Vector3.MoveTowards(\_cam.transform.position, \_cam.ScreenToWorldPoint(Input.mousePosition), cursorDistance \* scrollValue);

}

}

private Vector3 ClampCamera(Vector3 targetPosition)

{

float camHeight = \_cam.orthographicSize;

float camWidth = \_cam.orthographicSize \* \_cam.aspect;

float minX = \_mapMinX + camWidth;

float maxX = \_mapMaxX - camWidth;

float minY = \_mapMinY + camHeight;

float maxY = \_mapMaxY - camHeight;

float newX = Mathf.Clamp(targetPosition.x, minX, maxX);

float newY = Mathf.Clamp(targetPosition.y, minY, maxY);

return new Vector3 (newX, newY, targetPosition.z);

}

}

Barrel.cs

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

public class Barrel : StaticObject, IPointerDownHandler, IPointerUpHandler

{

[SerializeField] private Texture2D \_clickedCursorTexture;

[SerializeField] private BarellData \_barellData;

[SerializeField] private Text \_honeyText;

[SerializeField] private Sprite \_honeyBarrelSprite;

[SerializeField] private Sprite \_openedBarrelSprite;

private int \_honeyCapacity;

private int \_honeyOccupancy = 0;

public int HoneyCapacity

{

get { return \_honeyCapacity; }

}

private Hive \_hive;

private SpriteRenderer \_spriteRenderer;

private bool \_isOpened;

void Start()

{

SetBarellStats(\_barellData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

\_hive = FindObjectOfType<Hive>();

\_borderHint.SetActive(false);

\_isOpened = false;

\_nameText.text = \_name;

UpdateBarrelHintPanell();

\_hintPanel.SetActive(false);

}

void Update()

{

if (!\_isOpened)

{

SpriteRender();

}

}

public void AddHoney(int amount)

{

\_honeyOccupancy += amount;

if (\_honeyOccupancy > \_honeyCapacity)

{

\_honeyOccupancy = \_honeyCapacity;

}

UpdateBarrelHintPanell();

}

public void OnPointerDown(PointerEventData eventData)

{

\_isOpened = true;

\_borderHint.SetActive(false);

\_spriteRenderer.sprite = \_openedBarrelSprite;

Cursor.SetCursor(\_clickedCursorTexture, Vector2.zero, CursorMode.Auto);

}

public void OnPointerUp(PointerEventData eventData)

{

\_isOpened = false;

\_borderHint.SetActive(true);

Cursor.SetCursor(null, Vector2.zero, CursorMode.Auto);

}

public override void OnPointerClick(PointerEventData eventData)

{

if (\_honeyOccupancy > 0)

{

\_hive.AddHoney(\_honeyOccupancy);

\_honeyOccupancy = 0;

}

UpdateBarrelHintPanell();

\_hintPanel.SetActive(true);

}

public override void OnPointerEnter(PointerEventData eventData)

{

\_borderHint.SetActive(true);

\_hintPanel.SetActive(true);

}

public override void OnPointerExit(PointerEventData eventData)

{

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

private void SpriteRender()

{

if (\_honeyOccupancy > 0)

{

\_spriteRenderer.sprite = \_honeyBarrelSprite;

}

else

{

\_spriteRenderer.sprite = \_defaultSprite;

}

}

protected void UpdateBarrelHintPanell()

{

\_nameText.text = \_name;

\_honeyText.text = \_honeyOccupancy.ToString() + " / " + \_honeyCapacity.ToString();

}

private void SetBarellStats(BarellData barellData)

{

\_name = barellData.name;

\_honeyCapacity = barellData.honeyCapacity;

}

}

Flower.cs

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

public abstract class Flower : StaticObject

{

protected static int \_flowersCount = 0;

public static int FlowersCount

{

get { return \_flowersCount; }

set { \_flowersCount = value; }

}

public int PollenCount

{

get { return \_pollenCount; }

set { \_pollenCount = value; }

}

[SerializeField] protected Text \_pollenText;

protected int \_maxPollenCount;

protected int \_pollenCount;

protected void InitPollenCount()

{

\_pollenCount = Random.Range(1, \_maxPollenCount + 1);

}

protected void SetFlowerHintPanellSettings()

{

\_nameText.text = \_name;

string pollenName = \_pollenCount > 1 ? " НЕКТАРА" : " НЕКТАР";

\_pollenText.text = \_pollenCount.ToString() + pollenName;

}

public override void OnPointerClick(PointerEventData eventData)

{

\_borderHint.SetActive(true);

\_hintPanel.SetActive(true);

}

public override void OnPointerEnter(PointerEventData eventData)

{

\_borderHint.SetActive(true);

\_hintPanel.SetActive(true);

}

public override void OnPointerExit(PointerEventData eventData)

{

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

protected virtual void SetFlowerData(FlowerData FD)

{

\_name = FD.name;

\_maxPollenCount = FD.pollenAmount;

}

}

Chamomile.cs

using UnityEngine;

using UnityEngine.EventSystems;

public class Chamomile : Flower

{

[SerializeField] private FlowerData \_flowerData;

private SpriteRenderer \_spriteRenderer;

private void Start()

{

SetFlowerData(\_flowerData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

InitPollenCount();

SetFlowerHintPanellSettings();

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

}

Violet.cs

using UnityEngine;

using UnityEngine.EventSystems;

public class Violet : Flower

{

[SerializeField] private FlowerData \_flowerData;

private SpriteRenderer \_spriteRenderer;

private void Start()

{

SetFlowerData(\_flowerData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

InitPollenCount();

SetFlowerHintPanellSettings();

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

}

Sunflower.cs

using UnityEngine;

using UnityEngine.EventSystems;

public class Sunflower : Flower

{

[SerializeField] private FlowerData \_flowerData;

private SpriteRenderer \_spriteRenderer;

private void Start()

{

SetFlowerData(\_flowerData);

\_spriteRenderer = GetComponent<SpriteRenderer>();

InitPollenCount();

SetFlowerHintPanellSettings();

\_borderHint.SetActive(false);

\_hintPanel.SetActive(false);

}

}

BeeSpawner.cs

using UnityEngine;

public class BeeSpawner : MonoBehaviour

{

[SerializeField] private Bee \_pollinator;

[SerializeField] private Bee \_warrior;

[SerializeField] private Bee \_recycler;

[SerializeField] private int \_startPollinatorsAmount;

[SerializeField] private int \_startRecyclersAmount;

[SerializeField] private int \_startWarriorsAmount;

[SerializeField] private float \_spawnRadius = 1f;

private Vector2 \_spawnPos;

private void Start()

{

for (int i = 0; i < \_startPollinatorsAmount; ++i)

SpawnPollinator();

for (int i = 0; i < \_startRecyclersAmount; ++i)

SpawnRecycler();

for (int i = 0; i < \_startWarriorsAmount; ++i)

SpawnWarrior();

}

private Bee SpawnBee(Bee \_bee)

{

Bee.beeAmount += 1;

\_spawnPos = this.transform.position + Random.insideUnitSphere \* \_spawnRadius;

Bee bee = Instantiate(\_bee, \_spawnPos, Quaternion.identity);

bee.transform.SetParent(transform);

return bee;

}

public void SpawnPollinator()

{

BeePollinator.beePollinatorAmount += 1;

SpawnBee(\_pollinator);

}

public void SpawnWarrior()

{

BeeWarrior.beeWarriorAmount += 1;

SpawnBee(\_warrior);

}

public void SpawnRecycler()

{

BeeRecycler.beeRecyclerAmount += 1;

Barrel barrel = GetComponent<BarrelSpawner>().SpawnBarrel();

Bee beeRecycler = SpawnBee(\_recycler);

beeRecycler.GetComponent<BeeRecycler>().InitBarrel(barrel);

}

}

EnemySpawner.cs

using System;

using UnityEngine;

using UnityEngine.UI;

public class EnemySpawner : MonoBehaviour

{

[SerializeField] private ExitPanelController \_exitPanel;

[SerializeField] private WaspUpgrader \_waspUpgrader;

[SerializeField] private Text \_waveCountText;

[SerializeField] private Text \_waveTimerText;

[SerializeField] private float \_spawnRadiusX;

[SerializeField] private float \_spawnRadiusY;

[Range(0.1f, 5.0f)]

[SerializeField] private float \_waspSpawnDelay;

[Range(1.0f, 120.0f)]

[SerializeField] private float \_waveDelay;

[Serializable]

private class Wave

{

public Enemy enemy;

public int amount;

}

[SerializeField]

private Wave[] \_waves;

private Wave \_currentWave;

private float \_nextWaveTime = 0f;

private int \_currentWaveIndex = 0;

private float \_nextSpawnTime = 0f;

private int \_currentSpawnCount = 0;

private bool \_isPreparing = true;

private bool \_isWaveActive = false;

private float \_lastUpdateTime = 0f;

private void Start()

{

\_nextWaveTime = \_waveDelay;

}

private void Update()

{

float deltaTime = Time.deltaTime;

\_lastUpdateTime += deltaTime;

if (\_lastUpdateTime >= \_nextWaveTime && !\_isWaveActive)

{

\_waspUpgrader.Upgrade();

WaveProcess();

\_lastUpdateTime = 0f;

\_nextWaveTime = \_waveDelay;

}

if (\_isWaveActive)

{

\_nextWaveTime = \_waveDelay;

}

if (!\_isWaveActive)

{

Enemy[] enemies = FindObjectsOfType<Enemy>();

if (enemies.Length > 0)

{

\_nextWaveTime = \_waveDelay;

}

else

{

if (\_currentWaveIndex == \_waves.Length)

{

\_exitPanel.ShowResultPanel("ПОБЕДА!");

}

else

{

\_waveCountText.text = "ПОДГОТОВКА";

\_waveTimerText.enabled = true;

}

}

}

float timeLeft = Mathf.Clamp(\_nextWaveTime - \_lastUpdateTime, 0f, \_waveDelay);

\_waveTimerText.text = "СЛЕД. ВОЛНА ЧЕРЕЗ " + Mathf.CeilToInt(timeLeft).ToString() + " СЕК";

if (\_isPreparing)

{

\_waveCountText.text = "ПОДГОТОВКА";

\_waveTimerText.enabled = true;

}

if ((\_currentWaveIndex >= \_waves.Length - 1) && (\_isWaveActive))

{

\_waveCountText.text = "БОСС";

\_waveTimerText.enabled = false;

}

else if (\_isWaveActive)

{

\_waveCountText.text = "ВОЛНА " + (\_currentWaveIndex + 1).ToString();

\_waveTimerText.enabled = false;

}

}

private void WaveProcess()

{

\_currentWave = \_waves[\_currentWaveIndex];

\_currentSpawnCount = 0;

\_nextSpawnTime = \_lastUpdateTime;

\_isPreparing = false;

\_isWaveActive = true;

SpawnNextEnemy();

}

private void SpawnNextEnemy()

{

if (\_currentSpawnCount < \_currentWave.amount)

{

SpawnEnemy(\_currentWave.enemy);

\_currentSpawnCount++;

\_nextSpawnTime += \_waspSpawnDelay;

Invoke("SpawnNextEnemy", \_waspSpawnDelay);

}

else

{

\_currentWaveIndex++;

\_isWaveActive = false;

}

}

private void SpawnEnemy(Enemy enemy)

{

Vector2 spawnPosition = RandomEllipse(transform.position, \_spawnRadiusX, \_spawnRadiusY);

var obj = Instantiate(enemy.gameObject, spawnPosition, Quaternion.identity);

obj.transform.SetParent(transform);

}

private Vector2 RandomEllipse(Vector2 center, float radiusX, float radiusY)

{

float angle = UnityEngine.Random.value \* 360;

Vector2 position;

position.x = center.x + radiusX \* Mathf.Sin(angle \* Mathf.Deg2Rad);

position.y = center.y + radiusY \* Mathf.Cos(angle \* Mathf.Deg2Rad);

return position;

}

}

FlowerSpawner.cs

using UnityEngine;

public class FlowerSpawner : MonoBehaviour

{

[SerializeField] private Flower[] \_flowers;

[Range(1, 100)]

[SerializeField] private int \_startFlowerCount;

[Range(1, 500)]

[SerializeField] private int \_spawnLimit;

[Range(1f, 10f)]

[SerializeField] private float \_forbiddenSpawnRadius;

[Range(1f, 20f)]

[SerializeField] private float \_spawnRadiusX;

[Range(1f, 20f)]

[SerializeField] private float \_SpawnRadiusY;

[Range(0f, 10f)]

[SerializeField] private float \_spawnDelay;

private float \_nextSpawnTime = 0;

private int[] placementProbability = { 0, 1, 0, 0, 0, 1, 1, 0, 2, 0 };

private void Awake()

{

if (\_spawnLimit < \_startFlowerCount)

\_startFlowerCount = \_spawnLimit;

}

private void Start()

{

for (int i = 0; i < \_startFlowerCount;++i)

SpawnFlower();

Flower.FlowersCount = \_startFlowerCount;

}

private void Update()

{

if (Flower.FlowersCount < \_spawnLimit)

{

//InvokeRepeating("SpawnFlower", 0, \_spawnDelay);

\_nextSpawnTime += Time.deltaTime;

if (\_nextSpawnTime >= \_spawnDelay)

{

SpawnFlower();

\_nextSpawnTime = 0f;

}

}

}

private void SpawnFlower()

{

Flower flower = GetRandFlower();

Vector2 \_spawnPosition = RandomElipse(transform.position, \_spawnRadiusX, \_SpawnRadiusY);

var obj = Instantiate(flower, \_spawnPosition, Quaternion.identity);

obj.transform.SetParent(transform);

Flower.FlowersCount += 1;

}

private Flower GetRandFlower()

{

int randInx = placementProbability[Random.Range(0, placementProbability.Length)];

return \_flowers[randInx];

}

private Vector2 RandomElipse(Vector2 center, float radiusX, float radiusY)

{

float ang = Random.value \* 360;

float randX = Random.Range(\_forbiddenSpawnRadius, radiusX);

float randY = Random.Range(\_forbiddenSpawnRadius, radiusY);

Vector2 pos;

pos.x = center.x + randX \* Mathf.Sin(ang \* Mathf.Deg2Rad);

pos.y = center.y + randY \* Mathf.Cos(ang \* Mathf.Deg2Rad);

return pos;

}

}

BarrelSpawner.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BarrelSpawner : MonoBehaviour

{

[SerializeField] private Barrel \_barrel;

[SerializeField] private Transform \_spawnCenterPosition;

[Range(1.0f, 10.0f)]

[SerializeField] private float \_barrelSpawnRadius = 4.0f;

private void Start()

{

}

private void Update()

{

}

public Barrel SpawnBarrel()

{

Barrel barrel = Instantiate(\_barrel, RandomCircle(\_spawnCenterPosition.position), Quaternion.identity);

return barrel;

}

private Vector2 RandomCircle(Vector2 center)

{

float ang = Random.value \* 360;

Vector2 pos;

pos.x = center.x + \_barrelSpawnRadius \* Mathf.Sin(ang \* Mathf.Deg2Rad);

pos.y = center.y + \_barrelSpawnRadius \* Mathf.Cos(ang \* Mathf.Deg2Rad);

return pos;

}

}

Upgrader.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public abstract class Upgrader : MonoBehaviour, IUpgradable

{

protected int \_maxLevel;

protected int \_level;

public abstract void Upgrade();

}

WaspUpgrader.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WaspUpgrader : Upgrader

{

[SerializeField] private EnemyData \_data;

[SerializeField] private float \_gainHealthPoints;

[SerializeField] private float \_gainDamagePoints;

[SerializeField] private float \_gainFlightSpeed;

private void Start()

{

\_level = 1;

\_maxLevel = \_level;

}

public override void Upgrade()

{

\_level += 1;

\_maxLevel = \_level;

Wasp.MaxHealthPoints = (int)Mathf.Ceil(Wasp.MaxHealthPoints \* \_gainHealthPoints);

Wasp.FlightSpeed \*= \_gainFlightSpeed;

Wasp.damagePoints = (int)Mathf.Ceil(Wasp.damagePoints \* \_gainDamagePoints);

}

}

BeePollinatorUpgrader.cs

using System;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

public class BeePollinatorUpgrader : Upgrader

{

[SerializeField] private BeePollinatorData \_data;

[SerializeField] private BeePollinatorUIPanelController \_beePollinatorUIPanel;

[Serializable] public class BeePollinatorStats

{

public int maxHealthPoints;

public int maxSatietyPoints;

public float flightSpeed;

public float NCR;

public int nectarCapacity;

}

[SerializeField] private BeePollinatorStats[] \_upgradeStats;

[SerializeField] private int[] \_upgradePrices;

private BeePollinatorStats \_currentStats;

private void Start()

{

\_currentStats = new BeePollinatorStats();

\_currentStats.maxHealthPoints = \_data.healthPoints;

\_currentStats.maxSatietyPoints = \_data.satietyPoints;

\_currentStats.flightSpeed = \_data.flightSpeed;

\_currentStats.NCR = \_data.NCR;

\_currentStats.nectarCapacity = \_data.nectarCapacity;

\_maxLevel = \_upgradeStats.Length + 1;

\_level = 1;

\_beePollinatorUIPanel.UpdateBeePollinatorUIPanel(\_currentStats, \_level, GetUpgradePrice(\_level - 1));

}

private void Update()

{

if (\_level < \_maxLevel && Hive.HoneyOccupancy >= GetUpgradePrice(\_level - 1))

{

\_beePollinatorUIPanel.EnableUpgradeButton();

}

else

{

\_beePollinatorUIPanel.DisableUpgradeButton();

}

}

public override void Upgrade()

{

Hive.HoneyOccupancy -= GetUpgradePrice(\_level - 1);

\_level += 1;

\_currentStats = \_upgradeStats[\_level - 2];

BeePollinator.MaxHealthPoints = \_currentStats.maxHealthPoints;

BeePollinator.MaxSatietyPoints = \_currentStats.maxSatietyPoints;

BeePollinator.FlightSpeed = \_currentStats.flightSpeed;

BeePollinator.NCR = \_currentStats.NCR;

BeePollinator.NectarCapacity = \_currentStats.nectarCapacity;

\_beePollinatorUIPanel.UpdateBeePollinatorUIPanel(\_currentStats, \_level, GetUpgradePrice(Math.Min(\_level - 1, \_maxLevel - 2)));

if (\_level == \_maxLevel)

{

\_beePollinatorUIPanel.DisableUpgradeButton();

\_beePollinatorUIPanel.SetMaxUpgradeUI();

}

}

private int GetUpgradePrice(int level)

{

return \_upgradePrices[level];

}

}

BeeRecyclerUpgrader.cs

using System;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

public class BeeRecyclerUpgrader : Upgrader

{

[SerializeField] private BeeRecyclerData \_data;

[SerializeField] private BeeRecyclerUIPanelController \_beeRecyclerUIPanel;

[Serializable] public class BeeRecyclerStats

{

public int maxHealthPoints;

public int maxSatietyPoints;

public float flightSpeed;

public int productionEfficiency;

public float NPR;

}

[SerializeField] private BeeRecyclerStats[] \_upgrades;

[SerializeField] private int[] \_upgradePrices;

private BeeRecyclerStats \_currentStats;

private void Start()

{

\_currentStats = new BeeRecyclerStats();

\_currentStats.maxHealthPoints = \_data.healthPoints;

\_currentStats.maxSatietyPoints = \_data.satietyPoints;

\_currentStats.flightSpeed = \_data.flightSpeed;

\_currentStats.productionEfficiency = \_data.productionEfficiency;

\_currentStats.NPR = \_data.NPR;

\_maxLevel = \_upgrades.Length + 1;

\_level = 1;

\_beeRecyclerUIPanel.UpdateBeeRecyclerUIPanel(\_currentStats, \_level, GetUpgradePrice(\_level - 1));

}

private void Update()

{

if (\_level < \_maxLevel && Hive.HoneyOccupancy >= GetUpgradePrice(\_level - 1))

{

\_beeRecyclerUIPanel.EnableUpgradeButton();

}

else

{

\_beeRecyclerUIPanel.DisableUpgradeButton();

}

}

public override void Upgrade()

{

Hive.HoneyOccupancy -= GetUpgradePrice(\_level - 1);

\_level += 1;

\_currentStats = \_upgrades[\_level - 2];

BeeRecycler.MaxHealthPoints = \_currentStats.maxHealthPoints;

BeeRecycler.MaxSatietyPoints = \_currentStats.maxSatietyPoints;

BeeRecycler.FlightSpeed = \_currentStats.flightSpeed;

BeeRecycler.ProductionEfficiency = \_currentStats.productionEfficiency;

BeeRecycler.NPR = \_currentStats.NPR;

\_beeRecyclerUIPanel.UpdateBeeRecyclerUIPanel(\_currentStats, \_level, GetUpgradePrice(Math.Min(\_level - 1, \_maxLevel - 2)));

if (\_level == \_maxLevel)

{

\_beeRecyclerUIPanel.DisableUpgradeButton();

\_beeRecyclerUIPanel.SetMaxUpgradeUI();

}

}

private int GetUpgradePrice(int level)

{

return \_upgradePrices[level];

}

}

BeeWarriorUpgrader.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using static BeeRecyclerUpgrader;

public class BeeWarriorUpgrader : Upgrader

{

[SerializeField] private BeeWarriorData \_data;

[SerializeField] private BeeWarriorUIPanelController \_beeWarriorUIPanel;

[Serializable] public class BeeWarriorStats

{

public int maxHealthPoints;

public int maxSatietyPoints;

public float flightSpeed;

public int damagePoints;

public float detectionRange;

}

[SerializeField] private BeeWarriorStats[] \_upgrades;

[SerializeField] private int[] \_upgradePrices;

private BeeWarriorStats \_currentStats;

private void Start()

{

\_currentStats = new BeeWarriorStats();

\_currentStats.maxHealthPoints = \_data.healthPoints;

\_currentStats.maxSatietyPoints = \_data.satietyPoints;

\_currentStats.flightSpeed = \_data.flightSpeed;

\_currentStats.damagePoints = \_data.damagePoints;

\_currentStats.detectionRange = \_data.detectionRange;

\_maxLevel = \_upgrades.Length + 1;

\_level = 1;

\_beeWarriorUIPanel.UpdateBeeWarriorUIPanel(\_currentStats, \_level, GetUpgradePrice(\_level - 1));

}

private void Update()

{

if (\_level < \_maxLevel && Hive.HoneyOccupancy >= GetUpgradePrice(\_level - 1))

{

\_beeWarriorUIPanel.EnableUpgradeButton();

}

else

{

\_beeWarriorUIPanel.DisableUpgradeButton();

}

}

public override void Upgrade()

{

Hive.HoneyOccupancy -= GetUpgradePrice(\_level - 1);

\_level += 1;

\_currentStats = \_upgrades[\_level - 2];

BeeWarrior.MaxHealthPoints = \_currentStats.maxHealthPoints;

BeeWarrior.MaxSatietyPoints = \_currentStats.maxSatietyPoints;

BeeWarrior.FlightSpeed = \_currentStats.flightSpeed;

BeeWarrior.DamagePoints = \_currentStats.damagePoints;

BeeWarrior.DetectionRange = \_currentStats.detectionRange;

\_beeWarriorUIPanel.UpdateBeeWarriorUIPanel(\_currentStats, \_level, GetUpgradePrice(Math.Min(\_level - 1, \_maxLevel - 2)));

if (\_level == \_maxLevel)

{

\_beeWarriorUIPanel.DisableUpgradeButton();

\_beeWarriorUIPanel.SetMaxUpgradeUI();

}

}

private int GetUpgradePrice(int level)

{

return \_upgradePrices[level];

}

}

HiveUpgrader.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using static BeeRecyclerUpgrader;

public class HiveUpgrader : Upgrader

{

[SerializeField] private HiveData \_data;

[SerializeField] private HiveUIPanelController \_hiveUIPanel;

[Serializable]

public class HiveStats

{

public int maxIntegrityPoints;

public int beeCapacity;

public int nectarCapacity;

public int honeyCapacity;

}

[SerializeField] private HiveStats[] \_upgrades;

[SerializeField] private int[] \_upgradePrices;

private HiveStats \_currentStats;

void Start()

{

\_currentStats = new HiveStats();

\_currentStats.maxIntegrityPoints = \_data.maxIntegrityPoints;

\_currentStats.beeCapacity = \_data.beeCapacity;

\_currentStats.nectarCapacity = \_data.nectarCapacity;

\_currentStats.honeyCapacity = \_data.honeyCapacity;

\_maxLevel = \_upgrades.Length + 1;

\_level = 1;

\_hiveUIPanel.UpdateHiveUIPanel(\_currentStats, \_level, GetUpgradePrice(\_level - 1));

}

private void Update()

{

if (\_level < \_maxLevel && Hive.HoneyOccupancy >= GetUpgradePrice(\_level - 1))

{

\_hiveUIPanel.EnableUpgradeButton();

}

else

{

\_hiveUIPanel.DisableUpgradeButton();

}

}

public override void Upgrade()

{

Hive.HoneyOccupancy -= GetUpgradePrice(\_level - 1);

\_level += 1;

\_currentStats = \_upgrades[\_level - 2];

Hive.MaxIntegrityPoints = \_currentStats.maxIntegrityPoints;

Hive.BeeCapacity = \_currentStats.beeCapacity;

Hive.HoneyCapacity = \_currentStats.honeyCapacity;

Hive.NectarCapacity = \_currentStats.nectarCapacity;

Hive.CurrentIntegrityPoints = Hive.MaxIntegrityPoints;

\_hiveUIPanel.UpdateHiveUIPanel(\_currentStats, \_level, GetUpgradePrice(Math.Min(\_level - 1, \_maxLevel - 2)));

if (\_level == \_maxLevel)

{

\_hiveUIPanel.DisableUpgradeButton();

\_hiveUIPanel.SetMaxUpgradeUI();

}

}

private int GetUpgradePrice(int level)

{

return \_upgradePrices[level];

}

}

BeebuyingSystem.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public abstract class BeeBuyingSystem : MonoBehaviour, IBuyable

{

[SerializeField] protected BeeUIPanelController \_beePanel;

[SerializeField] protected BeeSpawner \_beeSpawner;

[SerializeField] protected int \_price;

[SerializeField] protected float \_inflationaryGrowth;

private void Start()

{

\_beePanel.UpdateBuyPriceUI(\_price);

}

protected void UpdateBuyButton()

{

if (Hive.BeeOccupancy < Hive.BeeCapacity && Hive.HoneyOccupancy >= \_price)

{

\_beePanel.EnableBuyButton();

\_beePanel.SetMaxBuyUI("КУПИТЬ");

}

else

{

if (Hive.BeeOccupancy == Hive.BeeCapacity)

{

\_beePanel.SetMaxBuyUI("МАКС.");

}

else

{

\_beePanel.SetMaxBuyUI("КУПИТЬ");

}

\_beePanel.DisableBuyButton();

}

}

public virtual void Buy()

{

Hive.HoneyOccupancy -= \_price;

UpdatePrice();

\_beePanel.UpdateBuyPriceUI(\_price);

}

protected virtual void UpdatePrice()

{

\_price = (int)(\_price \* \_inflationaryGrowth);

}

}

BeePollinatorBuyer.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BeePollinatorBuyer : BeeBuyingSystem

{

void Update()

{

UpdateBuyButton();

}

public override void Buy()

{

base.Buy();

\_beeSpawner.SpawnPollinator();

}

}

BeeRecyclerBuyer.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BeeRecyclerBuyer : BeeBuyingSystem

{

void Update()

{

UpdateBuyButton();

}

public override void Buy()

{

base.Buy();

\_beeSpawner.SpawnRecycler();

}

}

BeeWarriorBuyer.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BeeWarriorBuyer : BeeBuyingSystem

{

void Update()

{

UpdateBuyButton();

}

public override void Buy()

{

base.Buy();

\_beeSpawner.SpawnWarrior();

}

}

HiveUIPannelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class HiveUIPanelController : MonoBehaviour

{

[SerializeField] private Color \_maxValueColor;

[SerializeField] private Color \_defaultTextColor;

[SerializeField] private Text \_levelText;

[SerializeField] private Button \_upgradeButton;

[SerializeField] private Text \_upgradeButtonText;

[SerializeField] private Text \_upgradePriceText;

[SerializeField] private Text \_maxIntegrityPointsText;

[SerializeField] private Text \_beeCapacityText;

[SerializeField] private Text \_nectarCapacityText;

[SerializeField] private Text \_honeyCapacityText;

[SerializeField] private Button \_restoreButton;

[SerializeField] private Text \_restoreButtonText;

[SerializeField] private Text \_restorePriceText;

public virtual void DisableUpgradeButton()

{

\_upgradeButton.interactable = false;

}

public virtual void EnableUpgradeButton()

{

\_upgradeButton.interactable = true;

}

public void SetMaxUpgradeUI()

{

\_upgradeButtonText.text = "МАКС.";

\_upgradeButtonText.color = \_maxValueColor;

\_upgradePriceText.text = "-/-";

}

public void UpdateHiveUIPanel(HiveUpgrader.HiveStats stats, int level, int price)

{

\_levelText.text = "УРОВЕНЬ " + level.ToString();

\_upgradePriceText.text = price.ToString();

\_maxIntegrityPointsText.text = stats.maxIntegrityPoints.ToString();

\_beeCapacityText.text = stats.beeCapacity.ToString();

\_nectarCapacityText.text = stats.nectarCapacity.ToString();

\_honeyCapacityText.text = stats.honeyCapacity.ToString();

}

public void SetVoidRestore()

{

\_restoreButton.interactable = false;

\_restoreButtonText.text = "МАКС.";

}

public void EnableRestoreButton()

{

\_restoreButton.interactable = true;

\_restoreButtonText.text = "ПОЧИНИТЬ";

}

public void DisableRestoreButton()

{

\_restoreButton.interactable = false;

\_restoreButtonText.text = "ПОЧИНИТЬ";

}

public void UpdatePriceText(int price)

{

\_restorePriceText.text = price.ToString();

}

}

HiveBarsController.cs

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.UI;

public class HiveBarsController : MonoBehaviour

{

[SerializeField] private Hive \_hive;

[SerializeField] private Slider \_integritySlider;

[SerializeField] private Slider \_beeSlider;

[SerializeField] private Slider \_honeySlider;

[SerializeField] private Slider \_nectarSlider;

[SerializeField] private Text \_integrityText;

[SerializeField] private Text \_beeText;

[SerializeField] private Text \_honeyText;

[SerializeField] private Text \_nectarText;

void Update()

{

SetIntegritySlider();

SetBeeSlider();

SetHoneySlider();

SetNectarSlider();

}

private void SetIntegritySlider()

{

\_integritySlider.value = (float)Hive.CurrentIntegrityPoints / Hive.MaxIntegrityPoints;

\_integrityText.text = "ЦЕЛОСТНОСТЬ " + Hive.CurrentIntegrityPoints.ToString() + " / " + Hive.MaxIntegrityPoints.ToString();

}

private void SetBeeSlider()

{

\_beeSlider.value = (float)Hive.BeeOccupancy / Hive.BeeCapacity;

\_beeText.text = "ПЧЁЛЫ " + Hive.BeeOccupancy.ToString() + " / " + Hive.BeeCapacity.ToString();

}

private void SetHoneySlider()

{

\_honeySlider.value = (float)Hive.HoneyOccupancy / Hive.HoneyCapacity;

\_honeyText.text = "МЁД " + Hive.HoneyOccupancy.ToString() + " / " + Hive.HoneyCapacity.ToString();

}

private void SetNectarSlider()

{

\_nectarSlider.value = (float)Hive.NectarOccupancy / Hive.NectarCapacity;

\_nectarText.text = "НЕКТАР " + Hive.NectarOccupancy.ToString() + " / " + Hive.NectarCapacity.ToString();

}

}

ExitPanelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ExitPanelController : MonoBehaviour

{

[SerializeField] private GameObject \_panel;

[SerializeField] private Text \_titleText;

[SerializeField] private Button \_closeButton;

public void ShowPaneleByText(string text)

{

\_titleText.text = text;

\_panel.SetActive(true);

}

public void HidePanel()

{

\_panel.SetActive(false);

}

public void ShowResultPanel(string text)

{

Time.timeScale = 0;

\_titleText.text = text;

\_closeButton.interactable = false;

\_panel.SetActive(true);

}

}

BeeUIPanelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public abstract class BeeUIPanelController : MonoBehaviour

{

[SerializeField] protected Color \_indicatorMaxValue;

[SerializeField] protected Text \_levelText;

[SerializeField] protected Button \_upgradeButton;

[SerializeField] protected Text \_upgradeButtonText;

[SerializeField] protected Text \_upgradePriceText;

[SerializeField] protected Button \_buyButton;

[SerializeField] protected Text \_buyButtonText;

[SerializeField] protected Text \_buyPriceText;

[SerializeField] protected Text \_healthPointsText;

[SerializeField] protected Text \_satietyPointsText;

[SerializeField] protected Text \_flightSpeedText;

public virtual void DisableUpgradeButton()

{

\_upgradeButton.interactable = false;

}

public virtual void EnableUpgradeButton()

{

\_upgradeButton.interactable = true;

}

public virtual void DisableBuyButton()

{

\_buyButton.interactable = false;

}

public virtual void EnableBuyButton()

{

\_buyButton.interactable = true;

}

public void SetMaxUpgradeUI()

{

\_upgradeButtonText.text = "МАКС.";

\_upgradeButtonText.color = \_indicatorMaxValue;

\_upgradePriceText.text = "-/-";

}

public void SetMaxBuyUI(string text)

{

\_buyButtonText.text = text;

}

public virtual void UpdateUpgradePriceUI(int price)

{

\_upgradePriceText.text = price.ToString();

}

public virtual void UpdateBuyPriceUI(int price)

{

\_buyPriceText.text = price.ToString();

}

public virtual void UpdateLevelUI(int level)

{

\_levelText.text = "УРОВЕНЬ " + level.ToString();

}

}

BeeWarriorUIPanelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class BeeWarriorUIPanelController : BeeUIPanelController

{

[SerializeField] private Text \_damagePointsText;

[SerializeField] private Text \_detectionRangeText;

public void UpdateBeeWarriorUIPanel(BeeWarriorUpgrader.BeeWarriorStats stats, int level, int price)

{

UpdateLevelUI(level);

UpdateUpgradePriceUI(price);

\_healthPointsText.text = stats.maxHealthPoints.ToString();

\_satietyPointsText.text = stats.maxSatietyPoints.ToString();

\_flightSpeedText.text = stats.flightSpeed.ToString();

\_damagePointsText.text = stats.damagePoints.ToString();

\_detectionRangeText.text = stats.detectionRange.ToString();

}

}

BeerecyclerUIPanelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class BeeRecyclerUIPanelController : BeeUIPanelController

{

[SerializeField] private Text \_productionEfficiencyText;

[SerializeField] private Text \_NPRText;

public void UpdateBeeRecyclerUIPanel(BeeRecyclerUpgrader.BeeRecyclerStats stats, int level, int price)

{

UpdateLevelUI(level);

UpdateUpgradePriceUI(price);

\_healthPointsText.text = stats.maxHealthPoints.ToString();

\_satietyPointsText.text = stats.maxSatietyPoints.ToString();

\_flightSpeedText.text = stats.flightSpeed.ToString();

\_productionEfficiencyText.text = stats.productionEfficiency.ToString();

\_NPRText.text = stats.NPR.ToString();

}

}

BeePollinatorUIPanelController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class BeePollinatorUIPanelController : BeeUIPanelController

{

[SerializeField] private Text \_NCRText;

[SerializeField] private Text \_nectarCapacityText;

public void UpdateBeePollinatorUIPanel(BeePollinatorUpgrader.BeePollinatorStats stats, int level, int price)

{

UpdateLevelUI(level);

UpdateUpgradePriceUI(price);

\_healthPointsText.text = stats.maxHealthPoints.ToString();

\_satietyPointsText.text = stats.maxSatietyPoints.ToString();

\_flightSpeedText.text = stats.flightSpeed.ToString();

\_NCRText.text = stats.NCR.ToString();

\_nectarCapacityText.text = stats.nectarCapacity.ToString();

}

}

HiveData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "HiveData", menuName = "Static/Hive")]

public class HiveData : ScriptableObject

{

public string name;

public int maxIntegrityPoints;

public int beeCapacity;

public int nectarCapacity;

public int honeyCapacity;

}

FlowerData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName ="FlowerData", menuName = "Static/Flower")]

public class FlowerData : ScriptableObject

{

public string name;

public int pollenAmount;

}

EnemyData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "EnemyData", menuName = "Enemy/Enemy Data")]

public class EnemyData : ScriptableObject

{

public string name;

public int healthPoints;

public float flightSpeed;

public int damagePoints;

public float damageFrequency;

}

BeePollinatorData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "BeePollinatorData", menuName = "Bee/Bee Pollinator")]

public class BeePollinatorData : ScriptableObject

{

public string name;

public int healthPoints;

public int satietyPoints;

public float flightSpeed;

public float NCR;

public int nectarCapacity;

}

BeeRecyclerData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "BeeRecyclerData", menuName = "Bee/Bee Recycler")]

public class BeeRecyclerData : ScriptableObject

{

public string name;

public int healthPoints;

public int satietyPoints;

public float flightSpeed;

public float NPR;

public int productionEfficiency;

}

BeeWarrior.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "BeeWarriorData", menuName = "Bee/Bee Warrior")]

public class BeeWarriorData : ScriptableObject

{

public string name;

public int healthPoints;

public int satietyPoints;

public float flightSpeed;

public int damagePoints;

public float detectionRange;

}

BarrelData.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "BarellData", menuName = "Static/Barell")]

public class BarellData : ScriptableObject

{

public string name;

public int honeyCapacity;

}