Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по производственной практике

Выполнил:

студент группы 393551

Дегтярев А.А.

Руководитель от вуза:

Хотеев А.Л.

Руководитель от предприятия:

Щурук Д.А.

Минск 2017

Содержание

[Введение 3](#__RefHeading___Toc750_450552699)

[Индивидуальное задание 3](#__RefHeading___Toc752_450552699)

[Структура предприятия 4](#__RefHeading___Toc754_450552699)

[Сведения о программной платформе, проекте и инструментах 5](#__RefHeading___Toc756_450552699)

[Программная платформа 5](#__RefHeading___Toc758_450552699)

[Unity 5](#__RefHeading___Toc758_450552699)

[.Net Framework (Mono) 6](#__RefHeading___Toc758_450552699)

[SVN 7](#__RefHeading___Toc758_450552699)

Инструменты [7](#__RefHeading___Toc758_450552699)

[Внутреннее устройство и архитектура проекта 7](#__RefHeading___Toc758_450552699)

[Механизм работы с ресурсами 8](#__RefHeading___Toc758_450552699)

[Реализация индивидуального задания 9](#__RefHeading___Toc772_450552699)

[Заключение 12](#__RefHeading___Toc778_450552699)

[Список использованных материалов 13](#__RefHeading___Toc780_450552699)

[Приложение А 14](#__RefHeading___Toc782_450552699)

1. Введение
   1. Индивидуальное задание

Разработка модуля для сбора внутриигровой статистики.

**Требования к программе**

Необходимо реализовать модуль, который занимался бы сбором внутриигровой статистики и предоставлял внешний программный интерфейс (API), с помощью которого можно было бы получать количественные показатели срабатывания тех или иных внутриигровых событий (игрок совершил покупку товара, игрок построил комнату, игрок перезапустил уровень и т.д.). Основная программа написана на языке C#, использование внешних баз данных не предусмотрено, а вся статистика сохраняется через внутренний механизм работы с ресурсами в виде json файла.

**Задачи:**

* Ознакомиться с кодом и внутренним устройством/архитектурой проекта
* Ознакомиться с механизмом сохранения игры
* Реализация модуля сбора статистики.
* Проектирование программного интерфейса взаимодействия с модулем.
* Интегрирование модуля в проект.
* Тестирование.

* 1. Структура предприятия

ООО «Халфбас» - компания, занимающаяся разработки программного обеспечения (компьютерных игр); Компания занимается разработкой собственного продукта, под различные платформы: windows/macOS/linux/android/iOS.

Компания молодая и не имеет большого штата сотрудников, однако ее работники являются первоклассными специалистами с обширным опытом в области разработки игровых приложений (графики, игровой логики, ИИ).

Для разработки своих продуктов компания использует следующие технологии: Unity3d, Mono/.NET Framework, C#, SVN, C++.

Один из основных проектов компании игра **Basement**, доступна в интернет-магазине Steam.

Организация в своей деятельности руководствуется Законами Республики Беларусь, указами Президента Республики Беларусь, другими нормативными актами и уставом компании ООО «Халфбас»

Дата основания компании – 29 августа 2014.

1. Сведения о программной платформе, проекте и инструментах
   1. Программная платформа
   2. Unity

Unity – это кросс-платформенный игровой движок, разработанный компанией Unity Technologies. В основном, используется для разработки игр и симуляций для компьютеров, консолей и мобильных платформ. Сейчас поддерживается сборка на 27 различных платформ. Desktop ( Windows, Linux, MacOS ) , Consoles ( Xbox360, XboxOne, Playstation3, Playstation4, PS Vita, Nintendo Switch, Wii U, 3DS ) , Mobile ( iOS, Android ), Tv (Tizen, tvOS), WebGL и другие.

Unity – является движком общего назначения, который поддерживает как 2d так и 3d графику, скрипты на 3 различных языках(C#,JS,Boo), однако все они являются частью Mono Runtime(кроссплатформенная вариант .net framework). В последних версиях используется собственная runtime-библиотека, которая компилирует IL в C++ : IL2CPP. Она заметно ускорила работу внутриигровых скриптов.

Unity использует различные графические библиотеки в зависимости от целевой платформы. Direct3D и Vulkan для Windows, Xbox 360. OpenGL для Mac, Linux, Windows. OpenGL ES для Android/iOS, Metal для Mac/iOS/tvOS и патентованные графические библиотеки для консолей(Mantle/Switch).

**.NET Framework (Mono)**

Платформа .NET Framework — это фреймворк интегрированный в последние версии ОС Windows, поддерживает создание и выполнение приложений и веб-служб. Mono – является open-source копией платформы, и имеет компиляторы и среду выполнения для множества других платформ, отличных от Windows. Использования Mono в качестве среды выполнения кода обеспечило Unity возможность быстрого переноса кода проекта на любые платформы.

Двумя основными компонентами платформы .NET Framework являются общеязыковая среда выполнения (CLR) и библиотека классов .NET Framework. Основой платформы .NET Framework является среда CLR. Среду выполнения можно считать агентом/виртуальной машиной, которая управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом. Стоит отметить, что для некоторых платформ, чаще мобильных, используется AOT компиляция исходного кода, что ускоряет загрузку и исполнение.

Другой основной компонент платформы .NET Framework, библиотека классов, представляет полную объектно-ориентированную коллекцию типов, которые применяются для разработки приложений. Включая работу с датами, файловой системой, механизмов отражения типов и работы с бд. Unity использует уменьшенную версию стандартное библиотеки .net. Которую проще описать, как старый стандарт .net 2.0.

Язык C# является основным для разработки приложений для платформы .net. Boo и Javascript(unityscript) доступны для создания скриптов в Unity, однако они не получили широкого распространения, ибо все равно компилируются в IL, и не имеют особенностей Python/JS, а лишь копируют синтаксис.

**SVN**

В компании используется система контроля версий SVN. Это довольно популярное решение для игровых проектов, так как Git/HG довольно медленно работают с большими бинарными файлами. SVN является централизованной системой контроля версий, локальная копия постоянно синхронизируется с сервером. Я получил учетную запись и сохранял изменения в проекте используя SVN.

**Инструменты**

В качестве редактора кода я использовал Sublime Text. SVN – через командную строку. Движок Unity. Компьютер под управлением macOS.

**Архитектура приложения**

В составе проекта существует множество компонентов, отвечающих за разные части игровой логики. Глобально можно выделить несколько основных компонентов - это менеджер ресурсов (отвечает за загрузку ресурсов, сохранение, поддержки актуальности загруженных данных, так как игра может обновляться удаленно), менеджер игровых сцен (меню, редактор, игровая сцена). Последняя содержит множество блоков управляющих основной логикой, среди них мне нужно добавить модуль для сбора статистики.)

Сам движок предполагает использование **компонентной архитектуры**: все скрипты – являются небольшими компонентами, которые ассоциируются с игровыми объекты и инкапсулируют в себе какое-то поведение и данные. Сборщик статистики является **глобальным объектом**, он будет особым образом инициализироваться вместе с другими глобальными объектами-синглетонами. К числу таких объектов можно отнести: **AudioManager** – управляющий библиотекой звуков в игре, **InputManager** – для обработки различных сигналов с устройств ввода (джойстики, клавиатуры, мыши, тачскрин), **DialogManager** – особый компонент, выполняющий сборку диалогов из особого языка .ink. **TimeManager** – выступает счетчиком внутриигрового времени и планировщиком задач. **AIManager** – агрегирует управление над различными AI-компонентами. **Wallet/StorageManager** управляет данными о запасах различных внутриигровых ресурсов. **EventManager –** выполняет запуск и поддерживает исполнение внутриигровых сценариев.

**Механизм работы с ресурсами**

Игра подразумевает возможность изменения ресурсов игры самим игроком, вроде фоновых изображений, новых уровней и прочего UGC. Этот факт сильно определяет характер работы с игровыми ресурсами. Они могут быть изменены в процессе исполнения приложения и должны быть открыто доступны из ОС. Все настройки игры находятся в json-файлах в папке с игрой, изображения(спрайты) собираются во время исполнения в атласы и подгружаются в игру. Игра сохраняет текущее состояние сериализуя объекты в json.

1. Реализация индивидуального задания

Индивидуальное задание заключалось в необходимости реализации модуля для сбора внутриигровой статистики; События описываются строкой вроде “build\_room\_{room\_type}” и числовым параметром.

Предусматривалось несколько этапов:  
-Создание базового сборщика статистика  
-Предоставление программного интерфейса  
-Интеграция с игрой  
-Интеграция с игровым редактором  
-Тестирование

Основной класс представляющий событие:



Класс сериализумеых данных, которые будут записаны в json-файл



И статический класс предоставляющий интерфейс для сборки статистики   


Для сборки статистики, каждое событие выполняет вызов метода и передает данные о типе события и числовой параметр  
Ниже представлен пример сборки события, когда игрок снимает броню/оружие с бойца.



Также сборщик статистики позволяет регистрировать метод, который вызовется, когда какой-то событие зарегистрируется несколько раз registerTaskTracking. Это позволяет создавать задания, вроде «построй 5 комнат», «найми 3 работников» и т.д. Интеграция с редактором выполнена внутри редактора сценариев, который позволяет запускать другие элементы сценария, по срабатыванию заданного события. Задается оно в виде строки внутри редактора сценариев.

1. Заключение

В ходе производственной практики была изучена специфика предметной области, изучены основы работы с игровым движком, архитектура конкретного игрового проекта.

Было успешно выполнено индивидуальное задание, состоящее в реализации модуля для сбора внутриигровой статистики с интеграцией с игровым редактором. Данный модуль теперь активно используется в игре для создания квестов и заданий.

В ходе практики были получены навыки работы по специальности. Был повышен уровень владения вышеизложенными технологиями разработки программного обеспечения, благодаря интересному и познавательному индивидуальному заданию, а также благодаря хорошем и ответственным руководителям практики.

Список использованных материалов

[1] [Документация Unity3d](https://docs.unity3d.com/ScriptReference/) [Электронный ресурс].

[2] Microsoft Developer Network [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.msdn.microsoft.com>

[3] Алан Торн Искусство создания сценариев в Unity ДМК Пресс, 2016

[4] Фримен Э. Паттерны проектирования. – СПб.: Питер, 2011

Приложение А

Исходный код некоторых классов

|  |
| --- |
| using UnityEngine;  using Newtonsoft.Json;  using System.Collections.Generic;  using System;  public enum EventTrackingStrategy {  Add,  Max,  Update  }  [System.Serializable]  public class TrackedEvent {  public EventTrackingStrategy trackingStrategy;  public event Action<TrackedEvent> OnTrack;  public string name;  public int val;  public TrackedEvent(string name, int startval, EventTrackingStrategy strategy = EventTrackingStrategy.Add) {  this.name = name;  this.val = startval;  this.trackingStrategy = strategy;  }  public void Track(int val) {  switch (trackingStrategy) {  case EventTrackingStrategy.Add:  this.val += val;  break;  case EventTrackingStrategy.Max:  this.val = Mathf.Max(this.val, val);  break;  case EventTrackingStrategy.Update:  this.val = val;  break;  default:  throw new System.ArgumentOutOfRangeException();  }  if (OnTrack != null) {  OnTrack(this);  }  }  }  public class StatisticsTrackerData {  public Dictionary<string,TrackedEvent> registeredEvents;  [JsonIgnore]public Dictionary<string,List<EventObjectTrack>> registeredEventObjectTracks;  public StatisticsTrackerData() {  registeredEvents = new Dictionary<string,TrackedEvent>();  registeredEventObjectTracks = new Dictionary<string,List<EventObjectTrack>>();  }  }  public class StatisticsTracker {  static StatisticsTrackerData trackData;  public static void TrackEvent(string eventName, int val, EventTrackingStrategy strategy = EventTrackingStrategy.Add) {  Debug.Log("[Track::StatisticsTracker] tracked: " + eventName + " = " + val);  if (!trackData.registeredEvents.ContainsKey(eventName)) {  trackData.registeredEvents.Add(eventName, new TrackedEvent(eventName, val));  } else {  trackData.registeredEvents[eventName].Track(val);  }  TrackObjectivesProgress(eventName, val, strategy);  }  static void TrackObjectivesProgress(string eventName, int val, EventTrackingStrategy strategy) {  if (trackData.registeredEventObjectTracks.ContainsKey(eventName)) {  Debug.Log("[Track::StatisticsTracker] tracked objective: " + eventName + " = " + val);  for (int i = 0; i < trackData.registeredEventObjectTracks[eventName].Count; i++) {  trackData.registeredEventObjectTracks[eventName][i].trackedEvent.Track(val);  }  }  }  public static void RegisterTaskTracking(EventObjectTrack eventObjectTrack, bool usePrevTrack) {  Debug.Log("[Track::StatisticsTracker] registering task tracking: " + eventObjectTrack.trackedEventName);  if (!trackData.registeredEventObjectTracks.ContainsKey(eventObjectTrack.trackedEventName)) {  trackData.registeredEventObjectTracks[eventObjectTrack.trackedEventName] = new List<EventObjectTrack>();  }  trackData.registeredEventObjectTracks[eventObjectTrack.trackedEventName].Add(eventObjectTrack);  Debug.Log("[Track::StatisticsTracker] adding task tracking: " + eventObjectTrack.trackedEventName);  if (usePrevTrack) {  if (trackData.registeredEvents.ContainsKey(eventObjectTrack.trackedEventName)) {  eventObjectTrack.trackedEvent.Track(trackData.registeredEvents[eventObjectTrack.trackedEventName].val);  }  }  }  public static void DeregisterTaskTracking(EventObjectTrack eventObjectTrack) {  if (trackData.registeredEventObjectTracks.ContainsKey(eventObjectTrack.trackedEventName)) {  Debug.Log("[Track::StatisticsTracker] removing task tracking: " + eventObjectTrack.trackedEventName);  trackData.registeredEventObjectTracks[eventObjectTrack.trackedEventName].Remove(eventObjectTrack);  } else {  Debug.LogWarning("[Track::StatisticsTracker] no such task: " + eventObjectTrack.trackedEventName);  }  }  public static int GetEventOccurancies(string eventName) {  int count = 0;  if (trackData.registeredEvents.ContainsKey(eventName)) {  count += trackData.registeredEvents[eventName].val;  }  return count;  }  public StatisticsTracker(StatisticsTrackerData data = null) {  trackData = data ?? new StatisticsTrackerData();  }  public static StatisticsTrackerData GetPersistantData() {  return trackData;  }  } |

|  |
| --- |
| using System;  using UnityEngine;  public enum TaskTrackingStrategy {  MoreOrEqual,  More,  Less  }  public class EventObjectTrack: EventObject {  public string taskName;  public string trackedEventName;  public int trackedEventVal;  public EventTrackingStrategy trackedEventStrategy;  public TaskTrackingStrategy taskTrackingStrategy;  public bool useTime;  public bool usePreviousTrackRecords;  public bool showTaskNotification = true;  public bool showTaskProgress = true;  //runtime  public TrackedEvent trackedEvent;  public float time;  public event Action<TrackedEvent> eventTracked;  TaskNotificationModel task;  EventObject exitNode;  public EventObjectTrack() {  key = "Track";  exitNamesArray = new []{ "OnTracked", "OnTimeOver" };  exitNodes = new EventObject[exitNamesArray.Length];  }  protected override void OnEnter() {  base.OnEnter();  Location.Instance.Save(true);  Debug.Log("[Track::EventObjectTrack] tracking started: " + trackedEventName);  if (showTaskNotification) {  task = new TaskNotificationModel("checkbox", "ui\_help", this, time, false, NotificationStyle.Info, null, null, false);  NotificationManager.AddTaskNotification(task);  }  trackedEvent = new TrackedEvent(trackedEventName, 0, trackedEventStrategy);  trackedEvent.OnTrack += OnTrack;  StatisticsTracker.RegisterTaskTracking(this, usePreviousTrackRecords);  if (useTime) {  TimeManager.Instance.updateEvent += TimeOverUpdate;  }  }  private void TimeOverUpdate() {  time -= TimeManager.deltaTime;  if (time < 0f) {  OnTimeOver();  }  }  void OnTrack(TrackedEvent trackedEvent) {  if (eventTracked != null) {  eventTracked(trackedEvent);  }  switch (taskTrackingStrategy) {  case TaskTrackingStrategy.More:  if (trackedEvent.val > trackedEventVal) {  OnTrackComplete();  }  break;  case TaskTrackingStrategy.Less:  if (trackedEvent.val < trackedEventVal) {  OnTrackComplete();  }  break;  case TaskTrackingStrategy.MoreOrEqual:  if (trackedEvent.val >= trackedEventVal) {  OnTrackComplete();  }  break;  default:  throw new System.ArgumentOutOfRangeException();  }  }  public void OnTrackComplete() {  Debug.Log("[Track::EventObjectTrack] track complete: " + trackedEventName);  exitNode = exitNodes[0];  OnExit();  }  public void OnTimeOver() {  Debug.Log("[Track::EventObjectTrack] time over: " + trackedEventName);  exitNode = exitNodes[1];  OnExit();  }  public override void OnExit() {  base.OnExit();  trackedEvent.OnTrack -= OnTrack;  TimeManager.Instance.updateEvent -= TimeOverUpdate;  if (showTaskNotification) {  NotificationManager.RemoveInteractiveNotification(task);  }  StatisticsTracker.DeregisterTaskTracking(this);  exitNode.OnPreEnter();//call next node  }  public override EventObject GetNewInstance() {  return new EventObjectTrack();  }  public override void OnDestroy() {  base.OnDestroy();  TimeManager.Instance.updateEvent -= TimeOverUpdate;  if (showTaskNotification) {  NotificationManager.RemoveInteractiveNotification(task);  }  StatisticsTracker.DeregisterTaskTracking(this);  }  } |