Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по преддипломной практике

Выполнил:

студент группы 393551

Дегтярев А.А.

Руководитель от вуза:

Хотеев А.Л.

Руководитель от предприятия:

Щурук Д.А.

Минск 2018

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) 7](#__RefHeading___Toc758_450552699)

[SVN 8](#__RefHeading___Toc758_450552699)

Инструменты [8](#__RefHeading___Toc758_450552699)

[Внутреннее устройство и архитектура проекта 9](#__RefHeading___Toc758_450552699)

[Реализация индивидуального задания 10](#__RefHeading___Toc772_450552699)

[Заключение 19](#__RefHeading___Toc778_450552699)

[Список использованных материалов 20](#__RefHeading___Toc780_450552699)

1. Введение
   1. Индивидуальное задание

**Тема:** Разработка компонента внутриигрового редактора для генерации фоновых изображений, эффектов и декора.

**Требования к программе**

Необходимо реализовать модуль, который позволил бы игровым дизайнерам, занимающимся настройкой игры, быстро и с удобным интерфейсом, редактировать и создавать фоновые сцены для игровых уровней. Необходимо, чтобы это редактор был встроен в игру, являлся частью уже существующего игрового редактора, предоставлял внешний программный интерфейс (API), с помощью которого можно было бы программно конфигурировать фоны, имел возможность сохранения пресетов(шаблонов) в отдельный json-файл. Основная программа написана на языке C#, использование внешних баз данных не предусмотрено.

**Задачи:**

* Ознакомиться с кодом и внутренним устройством/архитектурой проекта
* Ознакомиться с работой редактора и системой хранения ресурсов
* Реализация модуля настройки фонов.
* Проектирование программного интерфейса взаимодействия с модулем.
* Интегрирование модуля в проект.
* Тестирование в составе системы.

* 1. Структура предприятия

ООО «Халфбас» - компания, занимающаяся разработки программного обеспечения (компьютерных игр), создана бывшими сотрудниками компании Nival; 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).

Активно движок используется как крупными разработчиками (Blizzard, EA, QuartSoft, Ubisoft), так и разработчиками Indie-игр (например, ремейк Мор. Утопия (Pathologic), Kerbal Space Program, Slender: The Eight Pages, Slender: The Arrival, Surgeon Simulator 2013, Baeklyse Apps: Guess the actor и т. п.) в силу наличия бесплатной версии, удобного интерфейса и простоты работы с движком.

Редактор Unity имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок поддерживает три сценарных языка: C#, JavaScript (модификация), Boo (диалект Python). Boo убран в 5-ой версии. Расчёты физики производит физический движок PhysX от NVIDIA.

Unity 3D поддерживает систему Level Of Detail (сокр. LOD), суть которой заключается в том, что на дальнем расстоянии от игрока высокодетализированные модели заменяются на менее детализированные, и наоборот, а также систему Occlusion culling, суть которой в том, что у объектов, не попадающих в поле зрения камеры не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект. При компиляции проекта создается исполняемый (.exe) файл игры (для Windows), а в отдельной папке — данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

Движок поддерживает множество популярных форматов, таких как:

.3ds, .max, .obj, .fbx, .dae, .ma, .mb, .blend для трёхмерных моделей;

.mp3, .ogg, .aiff, .wav, .mod, .it, .sm3 для звуковых файлов;

.psd, .jpg, .png, .gif, .bmp, .tga, .tiff, .iff, .pict, .dds для изображений;

.mov, .avi, .asf, .mpg, .mpeg, .mp4 для видеофайлов.

.txt, .htm, .html, .xml, .bytes для текста

Модели, звуки, текстуры, материалы, скрипты можно запаковывать в формат .unityassets и передавать другим разработчикам, или выкладывать в свободный доступ. Этот же формат используется во внутреннем магазине Unity Asset Store, в котором разработчики могут бесплатно и за деньги выкладывать в общий доступ различные элементы, нужные при создании игр. Чтобы использовать Unity Asset Store, необходимо иметь аккаунт разработчика Unity. Unity имеет все нужные компоненты для создания мультиплеера. Также можно использовать подходящий пользователю способ контроля версий. К примеру, Tortoise SVN или Source Gear.

**.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.

**Общая архитектура приложения**

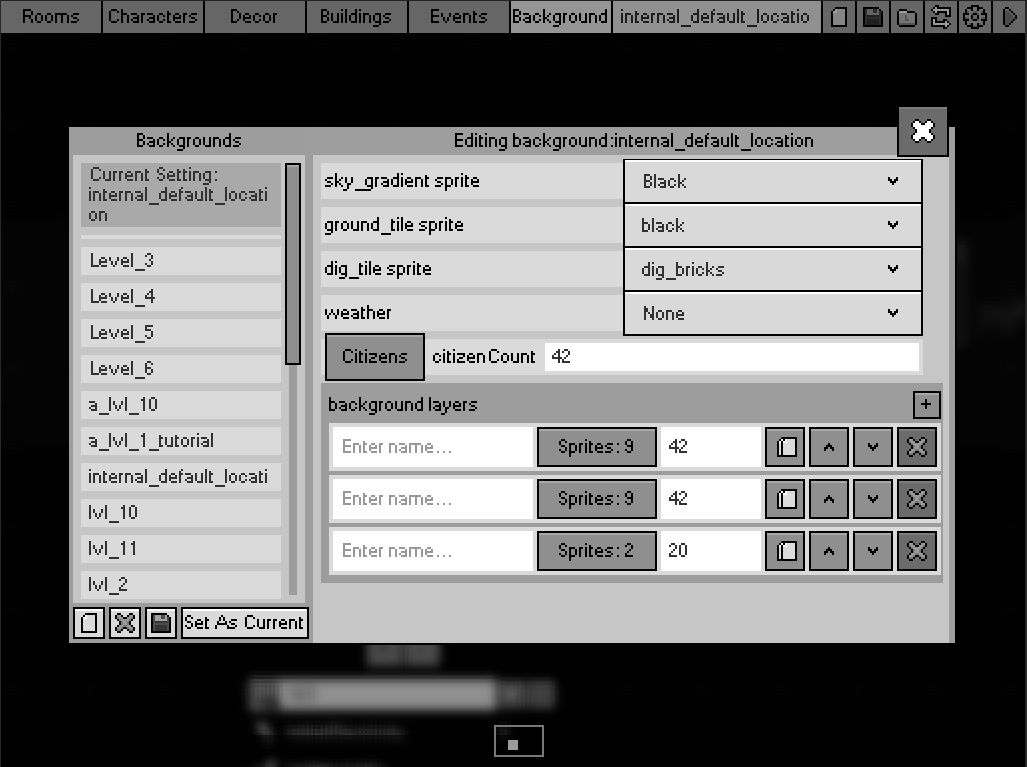
В составе проекта существует множество компонентов, отвечающих за разные части игровой логики. Можно выделить несколько основных компонентов - *менеджер ресурсов* (отвечает за загрузку ресурсов, сохранение, поддержки актуальности загруженных данных, так как игра может обновляться удаленно), *менеджер игровых сцен* (меню, редактор, игровая сцена), *редактор* представлен отдельной сценой.

Игровой движок предполагает использование **компонентной архитектуры**: все скрипты – являются небольшими компонентами, которые ассоциируются с игровыми объекты и инкапсулируют в себе какое-то поведение или данные. Модуль для редактирования фоновых изображений является составной частью редактора.

**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.

**Реализация индивидуального задания**

Ниже скриншот интерфейса редактора, с уже интегрированным модулем для редактирования фоновых изображений: 

Индивидуальное задание заключалось в реализации модуля для настройки фоновых изображений; необходимо было создать отдельное окно, в котором возможно было бы настраивать:

- Слои со списком фоновых изображений, с разными шансами их выпадения

- Префаб(шаблон) с эффектом погоды

- Фоновое изображение земли

- Фоновое изображения тайла для выкапывания

- Градиент, который будет использоваться для цвета тумана и неба

- Префабы фоновых персонажей и их число

Предусматривалось несколько этапов:  
- Создание интерфейса редактора  
- Предоставление программного интерфейса  
- Интеграция с игрой  
- Интеграция с игровым редактором

- Включение возможности сохранения в виде отдельного файла  
- Тестирование

Основной класс представляющий окно для настройки фонов:

Унаследован от BlockInteractionWindow – общего класса, который позволяет отображать окна во внутриигровом редакторе. Имеет ряд вспомогательных функций, вроде установки фокуса на объекте, поддержка команд джойстика, управлением вложенности окон.

|  |
| --- |
| using UnityEngine;  using System.Collections.Generic;  using UnityEngine.UI;  using Halfbus.UI;  public class WorldEditorBackgroundSettingsWindow : BlockInteractionWindow {  public static WorldEditorBackgroundSettingsWindow Instance;  [SerializeField] Text currentItemLabel;  [SerializeField] GameObject backgroundListItemPrefab;  [SerializeField] BaseContainer backgroundPresetListContainer;  [SerializeField] GameObject backgroundLayerContainerItem;  [SerializeField] BaseContainer backgroundLayerContainer;  [SerializeField] Dropdown skyGradientDropdown;  [SerializeField] Dropdown groundTileDropdown;  [SerializeField] Dropdown digTileDropdown;  [SerializeField] Dropdown weatherDropdown;  [SerializeField] Toggle myToggle;  [SerializeField] UIEditableField citizenCount;  [SerializeField] Text editingTitle;  bool isSetting;  List<string> weatherEffects;  LocationBackgroundSettings settings;  void Awake(){  Instance = this;  var weatherEffectKeys = new List<string>();  weatherEffectKeys.Add( "None" );  weatherEffectKeys.AddRange( Database<WeatherEffectNamePrefabPair>.GetInstance().GetAllKeys() );  weatherEffects = weatherEffectKeys;  backgroundPresetListContainer.OnItemClickAction += OnBackgroundItemClicked;  skyGradientDropdown.ClearOptions();  groundTileDropdown.ClearOptions();  digTileDropdown.ClearOptions();  weatherDropdown.ClearOptions();  backgroundLayerContainer.Clear();  skyGradientDropdown.AddOptions(Database<KeyGradientPair>.GetInstance().GetAllKeys());  groundTileDropdown.AddOptions(Database<NamedSprite>.GetInstance().GetAllKeys((s)=>{return s.subName == "GroundTiles";}));  digTileDropdown.AddOptions(Database<NamedSprite>.GetInstance().GetAllKeys((s)=>{return s.subName == "GroundDigTiles";}));  weatherDropdown.AddOptions(weatherEffects);  }  void ReloadBackgroundList() {  backgroundPresetListContainer.Clear();  foreach (var o in Database<LocationBackgroundSettings>.GetInstance().GetAll()) {  backgroundPresetListContainer.CreateItem(o, backgroundListItemPrefab);  }  }  public void SetAsCurrent(){  WorldEditor.Instance.currentLocationSettings.backgroundKey = settings.key;  currentItemLabel.text = "Current Setting: " + settings.key;  WorldEditor.Instance.layout.InitBackground();  }  void OnBackgroundItemClicked(object obj) {  settings = obj as LocationBackgroundSettings;  UpdateEditSettingsView();  }  public void SetupDefault(){  ReloadBackgroundList();  settings = WorldEditor.Instance.currentLocationSettings.background;  currentItemLabel.text = "Current Setting: " + settings.key;  UpdateEditSettingsView();  }  void UpdateEditSettingsView(){  isSetting = true;  editingTitle.text = "Editing background:" + settings.key;  skyGradientDropdown.value = Database<KeyGradientPair>.GetInstance().GetAllKeys().IndexOf(settings.skyGradientKey);  groundTileDropdown.value = Database<NamedSprite>.GetInstance().GetAllKeys((s)=>{return s.subName == "GroundTiles";}).IndexOf(settings.groundTile);  digTileDropdown.value = Database<NamedSprite>.GetInstance().GetAllKeys((s)=>{return s.subName == "GroundDigTiles";}).IndexOf(settings.groundDigTile);  weatherDropdown.value = weatherEffects.IndexOf(settings.weatherEffect);  citizenCount.Setup("citizenCount", settings.citizenCount.ToString(),OnApply,InputField.ContentType.IntegerNumber);  UpdateBackgroundLayersList();  isSetting = false;  }  public void OnNewBackgroundClicked(){  UIManager.Instance.ToggleWindow<TextInputPromptWindow>().SetupPrompt("Enter background name",  s => {  settings = new LocationBackgroundSettings();  settings.key = s;  StreamingAssetManager.Instance.SaveData(settings);  });  }  public void OnRemoveBackgroundClicked(){  if(!settings.isDefault){  UIManager.Instance.GetWindow<ModalWindow>().Show(  "Remove background settings",  "Are you sure you want to remove background settings?",  "Yes",  () => {  StreamingAssetManager.Instance.RemoveAsset( settings );  ReloadBackgroundList();  },  "No" );  }  }  public void CloseBackgroundSettingsWindow() {  UIManager.Instance.GetWindow<ModalWindow>().Show(  "Close background settings editor",  "Save changes?",  "Yes",  () => {  Save();  UIManager.Instance.CloseActiveWindow();  },  "No", () => {  UIManager.Instance.CloseActiveWindow();  } );  myToggle.isOn = false;  }  public void Save(bool silent = false){  if(settings!=null){  GameUtilities.Log("[BackgroundSettingsWindow] Saving");  StreamingAssetManager.Instance.SaveData(settings);  UIManager.Instance.GetWindow<ModalWindow>().Show(  "Background settings editor",  "Background: " + settings.key + " saved!",  "Ok", null );  }  if(!silent){  UIManager.Instance.CloseActiveWindow();  }  }  public void OnSetupCitizenListClicked() {  UIManager.Instance.OpenModalWindow<WorldEditorSettingSelectionWindow>().ListOptions(Database<CharacterData>.GetInstance().GetAllKeys((c) => {return c is CitizenData;}),  s => {settings.citizens = s != null ? s.ToList<string>() : null;}, true, settings.citizens);  }  public void OnApply(){  if(!isSetting){  settings.skyGradientKey = skyGradientDropdown.options[skyGradientDropdown.value].text;  settings.groundTile = groundTileDropdown.options[groundTileDropdown.value].text;  settings.groundDigTile = digTileDropdown.options[digTileDropdown.value].text;  settings.weatherEffect = weatherDropdown.options[weatherDropdown.value].text;  if(!string.IsNullOrEmpty(citizenCount.GetValue())) {  int tempInt;  settings.citizenCount = int.TryParse(citizenCount.GetValue(), out tempInt) ? tempInt : 0;  }  WorldEditor.Instance.layout.InitBackground();  }  }  //BACKGROUND LAYERS EDIT  public void UpdateBackgroundLayersList(){  backgroundLayerContainer.Clear();  foreach(var o in settings.layers){  backgroundLayerContainer.CreateItem(o, backgroundLayerContainerItem);  }  }  public void MoveUp(WorldEditorBackgroundLayerItem item) {  int index = backgroundLayerContainer.items.IndexOf(item);  if(index>0){  var data = settings.layers[index-1];  settings.layers[index-1]=settings.layers[index];  settings.layers[index] = data;  }  UpdateBackgroundLayersList();  }  public void MoveDown(WorldEditorBackgroundLayerItem item) {  int index = backgroundLayerContainer.items.IndexOf(item);  if(index<settings.layers.Count-1){  var data = settings.layers[index+1];  settings.layers[index+1]=settings.layers[index];  settings.layers[index] = data;  }  UpdateBackgroundLayersList();  }  public void Remove(WorldEditorBackgroundLayerItem item) {  int index = backgroundLayerContainer.items.IndexOf(item);  settings.layers.RemoveAt(index);  UpdateBackgroundLayersList();  }  public void Duplicate(WorldEditorBackgroundLayerItem item){  int index = backgroundLayerContainer.items.IndexOf(item);  settings.layers.Insert(index,settings.layers[index].Clone() as LocationBackgroundLayerSettings);  UpdateBackgroundLayersList();  }  public void OnAddLayerClicked(){  var backgroundLayer = new LocationBackgroundLayerSettings();  settings.layers.Add(backgroundLayer);  backgroundLayerContainer.CreateItem(backgroundLayer, backgroundLayerContainerItem);  }  } |

Класс сериализумеых данных, которые будут записаны в json-файл. Можно заметить что он не предоставляет никакого функционала, и является лишь структурой данных. Медот Clone – является обязательным для всех подклассов класса Settings, он используется для создания копий данных в ОЗУ, без лишних обращений к диску для I/O операций;

|  |
| --- |
| [System.Serializable]  public class LocationBackgroundSettings:StreamedSettings {  public List<LocationBackgroundLayerSettings> layers = new List<LocationBackgroundLayerSettings>();  public List<string> citizens;  public int citizenCount;  public string groundDigTile;  public string groundTile;  public string skyGradientKey;  public string weatherEffect;  }  [System.Serializable]  public class LocationBackgroundLayerSettings: Settings {  public List<string> sprites = new List<string>();  public int quantity = 10;  public int order;  public override object Clone() {  var clone = MemberwiseClone() as LocationBackgroundLayerSettings;  clone.sprites = sprites.Clone();  return clone;  }  } |

Класс представляющий элемент слоя фоновых объектов, стандартный класс, реализующий интерфейс IUIContainerItem, который позволяет настраивать внешний вид элементов списка:

|  |
| --- |
| using UnityEngine;  using System.Collections.Generic;  using UnityEngine.UI;  using Halfbus.UI;  public class WorldEditorBackgroundLayerItem : MonoBehaviour,IUIContainerItem {  [SerializeField] InputField textField;  [SerializeField] Text objectFieldLabel;  [SerializeField] InputField quantityFieldLabel;  LocationBackgroundLayerSettings backgroundLayerSettings;  public void Setup(object itemData, IUIContainer container) {  backgroundLayerSettings = itemData as LocationBackgroundLayerSettings;  UpdateState();  }  public void UpdateState() {  quantityFieldLabel.text = backgroundLayerSettings.quantity.ToString();  textField.text = backgroundLayerSettings.key;  objectFieldLabel.text = "Sprites: " + backgroundLayerSettings.sprites.Count;  }  public GameObject GetGameObject() {  return gameObject;  }  public void OnChangeName(string newName){  backgroundLayerSettings.key = newName;  UpdateState();  }  public void OnQuantityChanged(string min) {  int tempInt;  backgroundLayerSettings.quantity = int.TryParse(min, out tempInt) ? tempInt : 0;  UpdateState();  }  public void OnRemoveClicked(){  WorldEditorBackgroundSettingsWindow.Instance.Remove(this);  }  public void OnMoveUp(){  WorldEditorBackgroundSettingsWindow.Instance.MoveUp(this);  }  public void OnMoveDown(){  WorldEditorBackgroundSettingsWindow.Instance.MoveDown(this);  }  public void OnDuplicate(){  WorldEditorBackgroundSettingsWindow.Instance.Duplicate(this);  }  public void OnObjectPickClicked() {  UIManager.Instance.OpenModalWindow<WorldEditorSettingSelectionWindow>().ListOptions(Database<NamedSprite>.GetInstance().GetAllKeys((s) => {  return s.tag == "BackgroundImages" && s.subName != "GroundTiles" && s.subName != "GroundDigTiles";  }), s => {int tempInt;  backgroundLayerSettings.quantity = int.TryParse(quantityFieldLabel.text, out tempInt) ? tempInt : 0;  backgroundLayerSettings.sprites = s != null ? s.ToList<string>() : null;  WorldEditor.Instance.layout.InitBackground();  }, true, backgroundLayerSettings.sprites, true);  }  } |

Ниже представлен файл json с шаблоном фоновой сцены:

|  |
| --- |
| "$type": "LocationBackgroundSettings, Assembly-CSharp",  "layers": {  "$type": "System.Collections.Generic.List`1[[LocationBackgroundLayerSettings, Assembly-CSharp]], mscorlib",  "$values": [  {  "$type": "LocationBackgroundLayerSettings, Assembly-CSharp",  "sprites": {  "$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",  "$values": [  "building1\_medium",  "building2\_small",  "building3\_small",  "building4\_small",  "building5\_large"  ]  },  "quantity": 42,  "order": 0,  "key": null  },  {  "$type": "LocationBackgroundLayerSettings, Assembly-CSharp",  "sprites": {  "$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",  "$values": [  "building1\_medium",  "building2\_small",  "building3\_small",  "building4\_small",  "building5\_large"  ]  },  "quantity": 42,  "order": 1,  "key": null  },  {  "$type": "LocationBackgroundLayerSettings, Assembly-CSharp",  "sprites": {  "$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",  "$values": [  "building1\_medium",  "building2\_small",  "building3\_small",  "building5\_large",  "building4\_small"  ]  },  "quantity": 42,  "order": 2,  "key": null  }  ]  },  "citizens": {  "$type": "System.Collections.Generic.List`1[[System.String, mscorlib]], mscorlib",  "$values": [  "citizen1",  "citizen2",  "dog"  ]  },  "citizenCount": 42,  "groundDigTile": "dig\_bricks",  "groundTile": "clay",  "skyGradientKey": "Heat",  "weatherEffect": "None",  "isDefault": false,  "key": "a\_lvl\_10" |

1. Заключение

В ходе 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 редактирования фоновых сцен с интеграцией в игровой редактор. Данный модуль теперь активно используется в игре для создания уровней/игровых сцен.

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 хорошим 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

[5] Джозеф Хокинг Unity в действии. Мультиплатформенная разработка на C#