Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования*

Научно-исследовательская работа

На тему: «Разработка шаблона для создания проектов в жанре Role-Playing Game на Unreal Engine»

Студент Фёдоров Артемий Владиславович

Группа РК6-11М

Студент _____ **Фёдоров А.В.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Витюков Ф.А.**
подпись, дата *фамилия, и.о.*

Москва, 2025 г.

Содержание

Введение

Проекты в жанре Role Playing Game (Ролевая игра) — одни из самых старых в индустрии видеоигр. Современные игры этого жанра могут сильно различаться по визуальному оформление и геймплею, но классические RPG объединяет «математичность» игрового процесса, будь то тактическое планирование боев или расчёт характеристик персонажей.

Термин Top Down обозначает визуальный стиль, при котором игровое поле и персонажи видны сверху, часто с «высоты птичьего полёта». Этот стиль обычно используется в проектах, где игроку важно следить за глобальной ситуацией на игровом поле и контролировать несколько персонажей или сущностей одновременно.

Целью данной работы является исследование классических проектов в жанре Top Down RPG и разработка инструментов для создания подобных проектов.

Основные задачи исследования:

- Провести анализ существующих проектов с визуальной и игровой точки зрения.
- Изучить инструменты, предоставляемые движком Unreal Engine 4
- Разработать систему ортогональной камеры
- Разработать систему игрового поля, разделенные на клетки

Анализ существующих решений

В качестве объектов анализа были взяты коммерчески успешные видеоигровые проекты различных годов выпуска. Были измерены средние размеры клеток поля и размеры персонажей на экране в пикселях а также в сантиметрах. В качестве инструмента измерения физических метрик был взят монитор с диагональю 24 дюйма и соотношением сторон 16:9, что эквивалентно 51.1 см в ширину и 29.9 см в высоту.

Все полученные измерения представлены в таблице 1, скриншоты рассмотренны проектов представлены на рисунках 1-4.

Warcraft 2

Разрешение: 640x480

Год выпуска: 1995

В данном проекте была использована квадратая сетка с равными сторонами а также игровые персонажи и сущности, по размерами сопоставимые с одной игровой клеткой.

Рисунок 1. Скриншот видеоигры Warcraft 2

Majesty

Год выпуска: 2000

Разрешение: 800x600

В данном проекте игровая сетка не отображается напрямую, но влияет на расстановку внутриигровых зданий и перемещение персонажей по карте. Сетка была измерена с помощью измерения минимального возможного «шага» при выборе позиции нового здания.

Рисунок 2. Скриншот видеоигры Majesty

Fallout

Год выпуска: 1997

Оригинальное разрешение 640x480

Разрешение при измерении: 1280x720

Хотя все отображаемые здания и клетки «пола» имеют четырехугольную форму, в данном проекте используется шестиугольная сетка.

Рисунок 3. Скриншоты видеоигры Fallout

X-COM 2

Год выпуска: 2016

Разрешение: 1280x720

В отличие от остальных рассмотренных проектов, в данном используется не ортогональная камера, а камера с перспективой, то есть клетки и персонажи имеют разный размер на экране в зависмости от положения камеры. Для измерения были взяты клетки и персонажи, находящиеся в центре экрана.

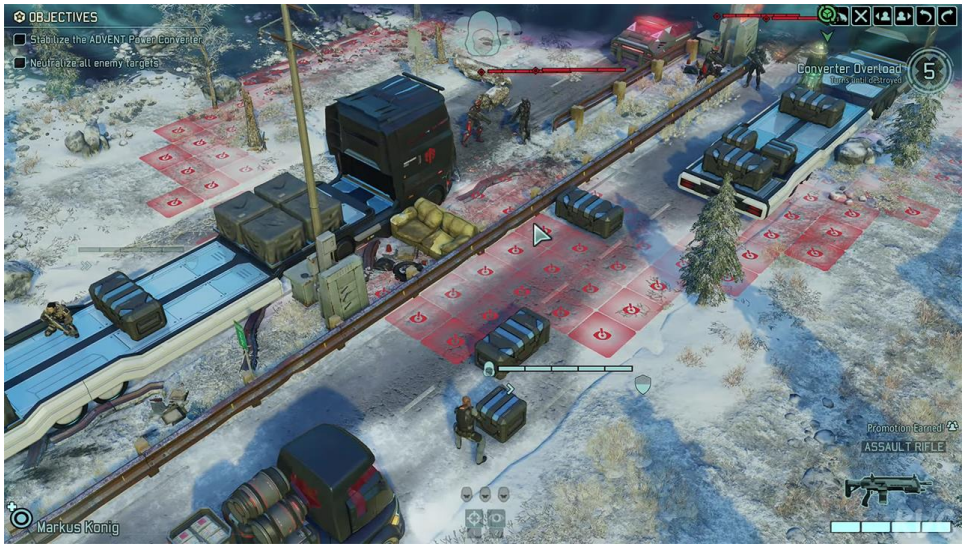

Рисунок 4. Скриншот видеоигры XCOM 2

Название	Год выпуска	Разрешение	Размер сетки в пикселях	Размер сетки в см	Размер персонажа в пикселях	Размер персонажа в см
Warcraft 2	1995	480x640	34x34	2.11x2.11	34x40	2.11x2.48
Majesty	2000	800x600	34x28	1.69x1.39	30x50	1.49x2.49
Fallout	1997	1280x720	66x35	2.63x1.45	47x88	1.87x3.65
XCOM 2	2016	1280x720	90x64	3,59x2,65	45x90	1.79x3.73

Таблица 1. Измерения размеров клеток и персонажей в различных проектах.

Полученные измерения будут использованы при дальнейшей разработке инструментов для создания схожих видеигровых проектов.

Настройка камеры

Трёхмерный движок Unreal Engine 4 предоставляет большой набор инструментов и расширяемых классов, удобных при разработке видеоигровых проектов.

Для настройки управления камерой был расширен класс APawn, отвечающий за игровых персонажей, которые могут быть контролируемы игроком. В класс-наследник BasicPawn были добавлены компоненты SpringArm (рычаг, на который крепится камера) и Camera. В настройках компонента Camera был выбран режим Orthographic, то есть режим отрисовки без перспективы.

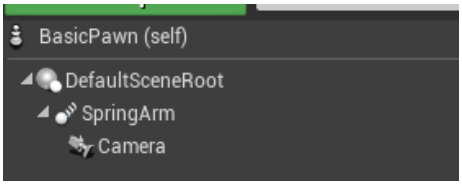

Рисунок 5. Структура класса BasicPawn

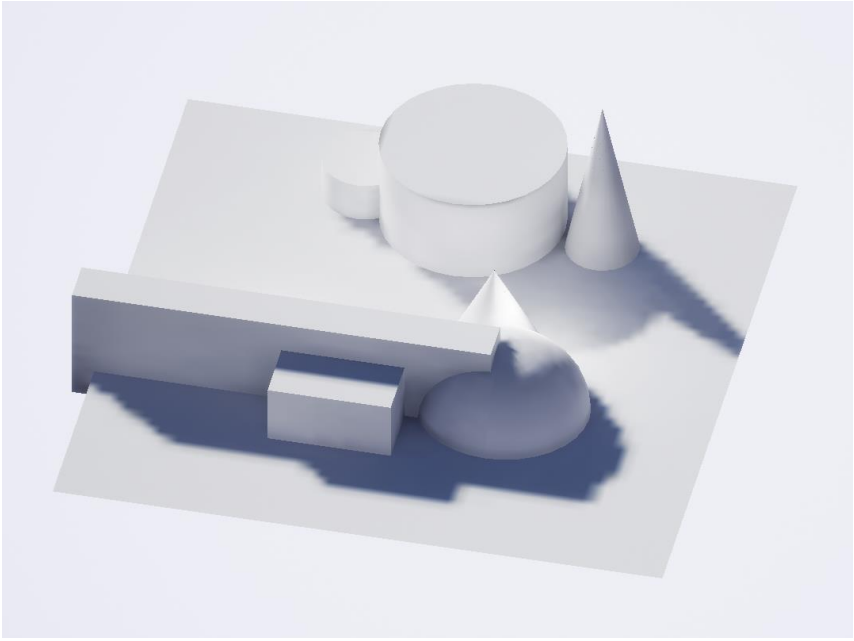

Рисунок 6 Набор геометрических фигур отображенный в ортографической проекции с помощью разработанной камеры

Для перемещения камеры на визуальном языке программирования Blueprints, предоставляемом движком Unreal Engine была реализована логика перемещения камеры в горизонтальной плоскости и также поворота вокруг своей оси в зависимости от ввода пользователя.

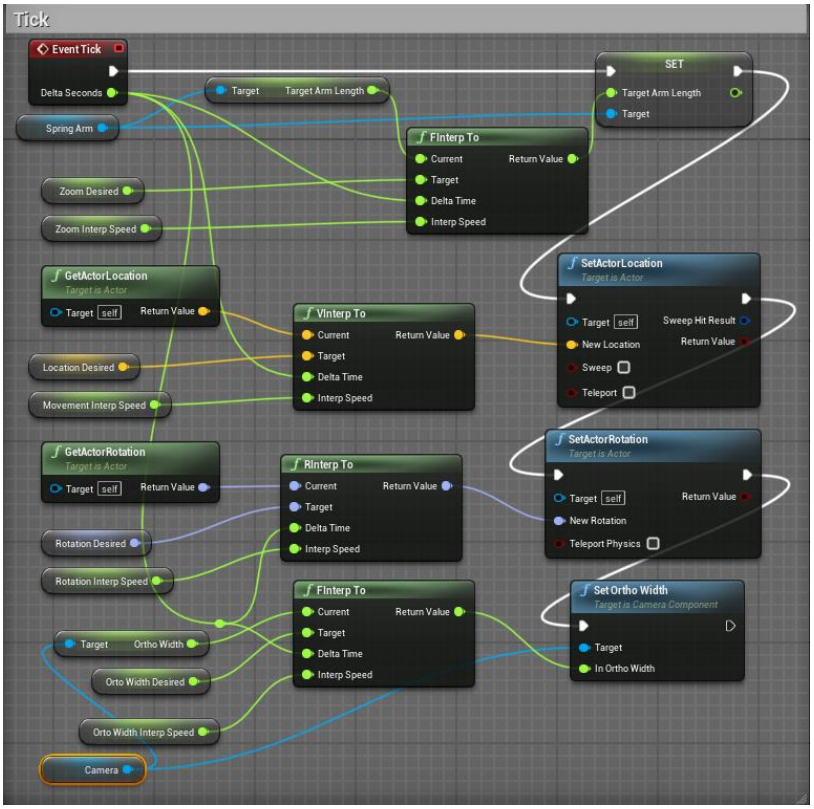

Рисунок 7. Blueprint – код, отвечающий за плавное перемещение камеры.

Разработка сетки поля

На языке C++ был разработан класс AGrid, наследующий от класса AActor, предоставляемого движком Unreal Engine. Разработанный класс включает в себя различные параметры сетки, такие как количество клеток и размер, а также тип сетки – шестиугольная или квадратная.

Листинг 1. Декларация класса AGrid

```
UCLASS()
class TOPDOWNRPG_API AGrid : public AActor
{
   GENERATED BODY()

public:
   AGrid();

protected:
   virtual void BeginPlay() override;

public:
   virtual void Tick(float DeltaTime) override;

   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   TArray<FTransform> Transforms;

   UFUNCTION(BlueprintCallable)
   void traceClick(FVector Location);
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   int SizeX = 1;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   int SizeY = 1;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float Scale = 1;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   bool IsHexGrid = false;
   virtual void OnConstruction(const FTransform& Transform) override;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   URuntimeVirtualTexture* GridRVT;
#pragma region SquareCell
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   UStaticMesh* SquareCell;

   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   UMaterialInstance* SquareMaterial;

   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float SquareScale = 1;
#pragma endregion
#pragma region HexCell
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   UStaticMesh* HexCell;

   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   UMaterialInstance* HexMaterial;

   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float HexScale = 1;
#pragma endregion
#pragma region CellMeshes
   UPROPERTY(VisibleAnywhere)
   UInstancedStaticMeshComponent* CellMeshes;
#pragma endregion
};
```

На языке C++ был разработан класс AGrid, наследующий от класса AActor, предоставляемого движком Unreal Engine. Разработанный класс включает в себя различные параметры сетки, такие как количество клеток и размер, а также тип сетки – шестиугольная. Было настроено создание сетки: размещение моделей отдельных клеток в зависмости от типа сетки.

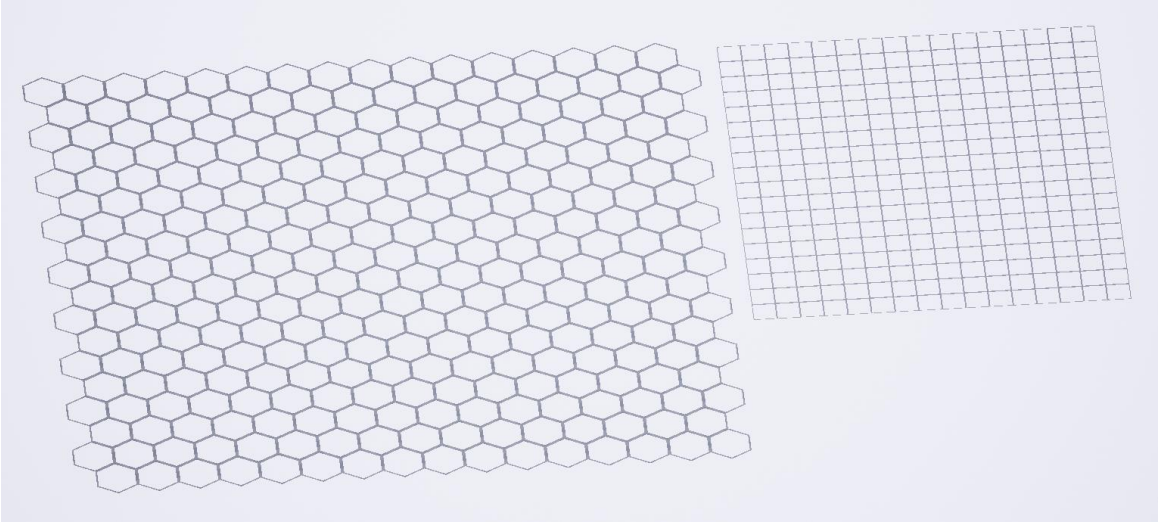

Рисунок 8. Сетки, состоящие из отдельных 3д моделей клеток.

Листинг 2. Расстановка клеток сетки в зависмости от типа сетки.

```
void AGrid::OnConstruction(const FTransform& Transform)
{
   Super::OnConstruction(Transform);
   CellMeshes->RuntimeVirtualTextures.Add(GridRVT);
   if (IsHexGrid)
   {
      CellMeshes->SetStaticMesh(HexCell);
      CellMeshes->SetMaterial(0, HexMaterial);
   }
   else
   {
      CellMeshes->SetStaticMesh(SquareCell);
      CellMeshes->SetMaterial(0, SquareMaterial);
   }

   CellMeshes->ClearInstances();
   CellMeshes->NumCustomDataFloats=1;
   for (int Index1 = 0; Index1 < SizeX; Index1++)
   {
      for (int Index2 = 0; Index2 < SizeY; Index2++)
      {
         FVector Translation, CScale;
         if (IsHexGrid)
         {
            if(Index1%2)
            {
               Translation = FVector(Scale * 1.5 * Index1,
                             Scale * sqrt(3) * Index2,
                             -100);
            }
            else
            {
               Translation = FVector(Scale * 1.5 * Index1,
```

```
                            Scale * sqrt(3)/2+Scale * sqrt(3) * Index2,
                            -100);
                }
                CScale = FVector(HexScale*Scale);
            }
            else
            {
                Translation = FVector(Scale * Index1,
                                      Scale * Index2,
                                      -100);
                CScale = FVector(SquareScale*Scale);
            }
            FQuat Rotation = FQuat(0.0f, 0.0f, 0.0f, 1.0f);
            Rotation.Normalize();
            FTransform InstanceTransform = FTransform(Rotation, Translation, CScale);
            CellMeshes->AddInstance(InstanceTransform);
            if(Index1%2 && Index2%2)
            {
                CellMeshes->SetCustomDataValue(CellMeshes->GetInstanceCount()-1, 0, 0.0f, true);
            }else
            {
                CellMeshes->SetCustomDataValue(CellMeshes->GetInstanceCount()-1, 0, 0.0f, true);
            }
        }
    }
}
```

С помощью технологии Runtime Virtual Texture, предоставляемой Unreal Engine и позволяющей записывать данные в текстуры в реальном времени была сделана проекция разработанной сетки на произвольные поверхности, что позволит использовать игровые поля с переменной высотой.

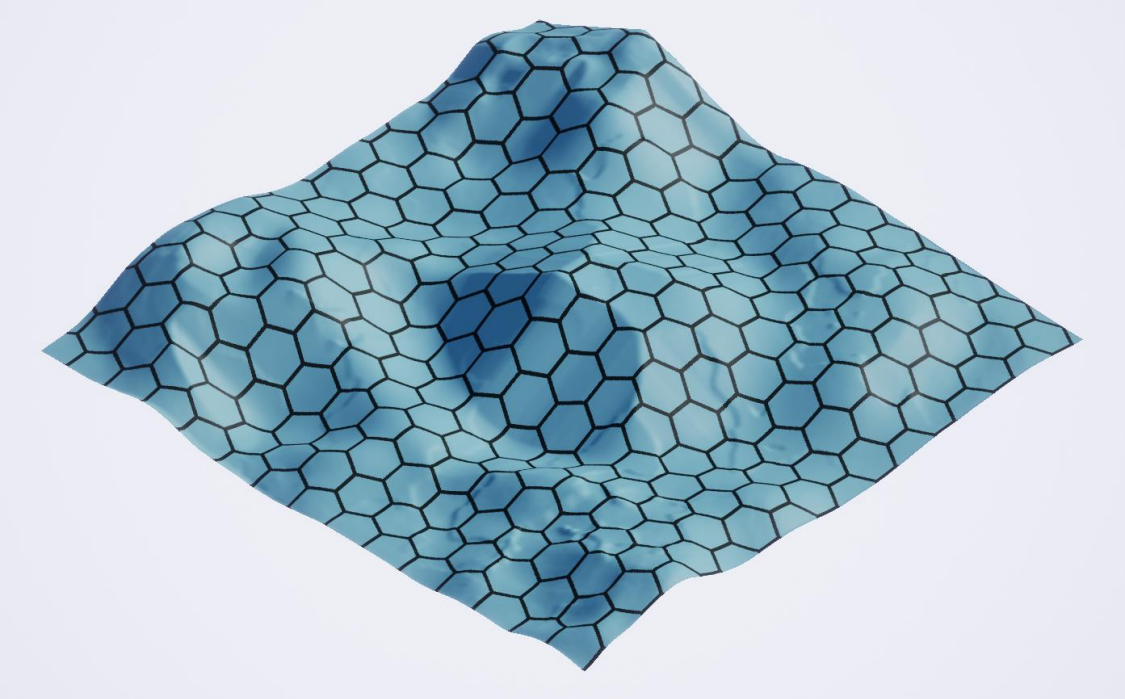

Рисунок 9. Проекция разработанной сетки на ландшафт произвольной формы

Для возможности взаимодействия пользователя с разработанной сеткой была настроена система отклика на нажатия кнопки мыши. При нажати ЛКМ просходит трассировка лучом, определяющая на какую ячейку было произведено нажатие. После чего выделенная ячейка окрашивается в определенный цвет.

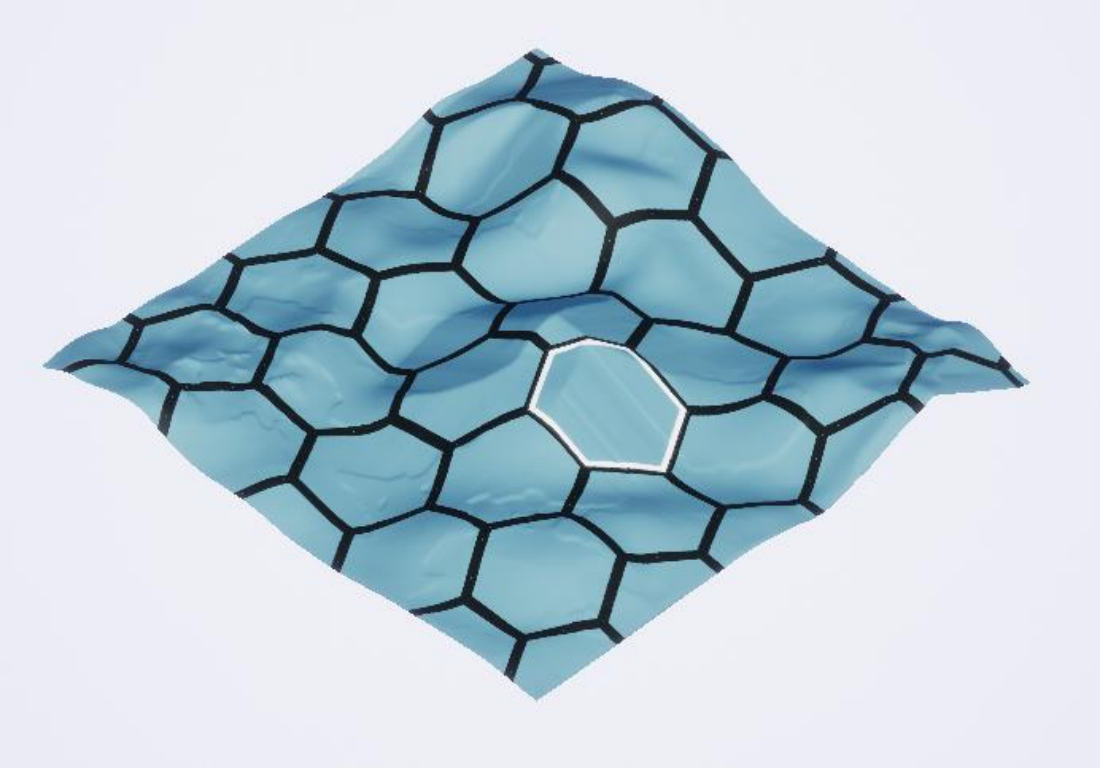

Рисунок 10. Шестиугольная сетка с одной клеткой, выделенной нажатием левой кнопки мыши.

Заключение

В ходе выполнения данной научно-исследовательской работы была положена основа набора инструментов для создания видеоигр в стиле Top Down RPG и были выполнены следующие задачи:

- Проведен анализ существующих проектов с визуальной и точки зрения.
- Изучены инструменты, предоставляемые движком Unreal Engine 4
- Разработана система ортогональной камеры с возможность перемещения и поворота.
- Разработана систему визуализации игрового поля, разделенного на клетки с возможностью отображения клеток на произвольной поверхности.

Список литературы

1. Unreal Engine 4 Documentation // Unreal Engine Documentation URL: https://docs.unrealengine.com/. Дата обращения: 11.10.2024.
2. Божко А.Н., Жук Д.М., Маничев В.Б. Компьютерная графика. [Электронный ресурс] // Учебное пособие для вузов. − М.: Изд-во МГТУ им. Н. Э. Баумана, 2007. - 389 с., - ISBN 978-5-7038-3015-4, Режим доступа: http://ebooks.bmstu.ru/catalog/55/book1141.html. Дата обращения: 12.10.2024;
3. Programming Quick Start // Unreal Engine Documentation URL: https://docs.unrealengine.com/5.0/en-US/unreal-engine-cpp-quick-start/. Дата обращения: 29.10.2023.
4. Geometry instancing // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Geometry_instancing. Дата обращения: 14.10.2024.

5. Modeling – Blender Manual // Blender Manual URL: https://docs.blender.org/manual/en/latest/modeling/index.html. Дата обращения: 01.12.2024.