МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

СОГЛАСОВАНО
Руководитель ОП

_____ Е.В. Пустовалов

«_24__» __июня___2018 г.

«УТВЕРЖДАЮ»
Проректор по развитию

_____ Д.И. Земцов

«_27__» __июня___2018 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ «ТЕХНОЛОГИИ ВИРТУАЛЬНОЙ И ДОПОЛНЕННОЙ РЕАЛЬНОСТИ В ПРОМЫШЛЕННОМ ПРОИЗВОДСТВЕ»

направления 09.04.01 Информатика и вычислительная техника
Магистерская программа «Технологии виртуальной и дополненной реальности»
Форма подготовки очная

курс 2 семестр 3,4
лекции 18 час.
практические занятия 0 час.
лабораторные работы 54 час.
всего часов аудиторной нагрузки 72 час.
самостоятельная работа 144 час.
контрольные работы программой не предусмотрены
курсовая работа/проект – не предусмотрено
зачет с оценкой– 3,4 семестр
экзамен - не предусмотрено учебным планом

Рабочая программа составлена в соответствии с требованиями федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.04.01 – Информатика и вычислительная техника, утвержденного приказом Министерства образования и науки Российской Федерации от 30.10.2014 № 1420

Рабочая программа рассмотрена и утверждена на заседании Дирекции Школы цифровой экономики 24 июня 2018 г., протокол №2

Составитель(и): к.т.н. Змеу В.К.; асс. Спорышев М.С.

Оборотная сторона титульного листа РПД

I. Рабочая программа пересмотрена на заседании Дирекции Школы цифровой экономики:
Протокол от «_____» _____ 20___ г. № _____

Заместитель директора ШЦЭ
по учебной и воспитательной работе _____ _____
(подпись) (И.О. Фамилия)

II. Рабочая программа пересмотрена на заседании Дирекции Школы цифровой экономики:
Протокол от «_____» _____ 20___ г. № _____

Заместитель директора ШЦЭ
по учебной и воспитательной работе _____ _____
(подпись) (И.О. Фамилия)

АННОТАЦИЯ

Рабочая программа учебной дисциплины «Технологии виртуальной и дополненной реальности в промышленном производстве» предназначена для студентов, обучающихся по направлению подготовки 09.04.01 Информатика и вычислительная техника (уровень магистратуры), профиль «Технологии виртуальной и дополненной реальности».

Рабочая программа разработана на основе макета рабочей программы учебной дисциплины для образовательных программ высшего образования – программ бакалавриата, специалитета, магистратуры ДВФУ, утверждённого приказом ректора ДВФУ от 08.05.2015 № 12-13-824.

Дисциплина «Технологии виртуальной и дополненной реальности в промышленном производстве» входит в вариативную часть блока «Дисциплины (модули) по выбору» (Б1.В.ДВ.02) учебного плана подготовки магистров.

Общая трудоемкость освоения дисциплины составляет 6 зачетных единиц или 216 часов. Дисциплина реализуется на 2 курсе в 3 и 4 семестре.

Семестр	Аудиторные занятия			Самостоя-тельная работа	Контроль	Всего по дисциплине	
	Лекции	Лабораторные работы	Всего			Часы	Зачетные единицы
3 семестр	–	36	36	72	Зачет оценкой	108	3
4 семестр	18	18	36	72	Зачет оценкой	108	3
Всего	**18**	**54**	**72**	**144**		**216**	**6**

Промышленные предприятия вынуждены осваивать технологии виртуальной реальности, если хотят вписаться в историю «Индустрии 4.0.», встать на путь цифровизации и полной автоматизации производственных процессов. «Интернет вещей», VR и AR – важные составные части новой организации бизнес-моделей.

Цель изучения дисциплины – изучение. технологий и алгоритмов виртуальной (VR) и дополненной реальности (AR); получение практических навыков по применению виртуальной (VR) и дополненной реальности (AR) в промышленном производстве

Задачи:

изучение основных технологий и получение практических навыков в области создания приложений VR и AR для промышленности:

- дизайн и прототипирование (дизайн новых продуктов) – создание голографических моделей в реальном времени с целью симулировать разные варианты использования, примерять различные сценарии применения продукта и предсказывать потенциальные ошибки дизайна;
- сборка продукции – технологии «смешанной» реальности могут значительно облегчить сборку конечной продукции, состоящей из множества элементов;
- проверка качества – технологии дополненной реальности помогут упростить сложный и отнимающий много времени процесс проверки качества продукции на заводе-изготовителе;
- сервисное обслуживание – технологии, помогающие в реальном времени проводить ремонт и замену аппаратов;
- виртуальные тренажеры – самый востребованный на сегодня вариант применения VR в промышленности – обучение персонала (операторов и специалистов поддержки).

Для прохождения курса потребуются базовые знания линейной алгебры, дифференциального исчисления, языков программирования: C/C++ 11 или новее и/или Python.

В результате данной дисциплины у обучающихся формируются следующие профессиональные компетенции (элементы компетенций).

Код и формулировка компетенции	**Этапы формирования компетенции**	
ПК-8 – способность проектировать распределенные информационные системы,	Знает	- методы проектирования распределенных информационных систем
	Умеет	- проектировать распределенные информационные системы, их компоненты и

их компоненты и протоколы их взаимодействия		протоколы их взаимодействия
	Владеет	- навыками проектирования распределенных систем, их компонентов и протоколов их взаимодействия
ПК-13 – способность к программной реализации распределенных информационных систем	Знает	- основные стандарты в области организации доступа к распределенным информационным системам; - основные технологии реализации распределенных систем; - основные технологии поиска информации в распределенных информационных системах; - основные технологии представления и передачи структурированной информации в распределенных информационных системах
	Умеет	- проектировать распределенные информационные системы; - разрабатывать серверное и клиентское программное обеспечения распределенных информационных систем; - пользоваться архивами свободно распространяемого программного обеспечения; - конструировать программные комплексы для распределенных информационных систем; - организовывать преобразование данных на основе стандартных технологий; - создавать пользовательские интерфейсы для доступа к распределенным информационным системам.
	Владеет	навыками: - программной реализации распределенных информационных систем; - конструирования программных комплексов для распределенных информационных систем; - создания пользовательских интерфейсов для доступа к распределенным информационным системам.
ПК-16 – способность к созданию служб сетевых протоколов	Знает	- сетевые технологии и протоколы; - принципы построения сетевого взаимодействия
	Умеет	- программировать службы сетевых протоколов
	Владеет	- навыками создания служб сетевых протоколов
ПК-19 – способность к применению современных технологий разработки программных комплексов с использованием CASE-средств, контролировать качество разрабатываемых программных продуктов	Знает	- современные средства автоматизации процесса разработки информационных систем и программного обеспечения (CASE-средства); - основные функциональные, технические и эксплуатационные характеристики, предъявляемые к разрабатываемым программным продуктам
	Умеет	- разрабатывать программные комплексы с использованием современных CASE-средств; - контролировать качество разрабатываемых

		программных продуктов
	Владеет	- навыками использования CASE-средств; - методами анализа и повышения качества программных продуктов

Для формирования вышеуказанных компетенций в рамках дисциплины «Технологии виртуальной и дополненной реальности в промышленном производстве» применяются следующие методы активного/интерактивного обучения: лекция-беседа, метод автоматизированного обучения.

При выполнении различных видов работ используются следующие технологии:

1. *Проблемное обучение* – стимулирование обучающихся к самостоятельному приобретению знаний, необходимых для решения конкретной проблемы.

2. *Контекстное обучение* – мотивация студентов к усвоению знаний путём выявление связей между конкретным знанием и его применением.

3. *Обучение на основе опыта* – активизация познавательной деятельности обучающихся за счёт ассоциации и собственного опыта с предметом обучения.

I. СТРУКТУРА И СОДЕРЖАНИЕ ТЕОРЕТИЧЕСКОЙ ЧАСТИ КУРСА

4 семестр

Лекции (18 часов)

Тема 1. Основы технологий виртуальной и дополненной реальности Базовые понятия и определения технологий виртуальной и расширенной реальности. Функциональные возможности современных приложений и сред с иммерсивным контентом. Сферы применения и использования технологий виртуальной и расширенной реальности. Составляющие иммерсивного контента. Идея и сценарий для приложений разного уровня погружения в виртуальное пространство.	**2 часа**
Тема 2. Устройства визуализации и взаимодействия для иммерсивных сред Классификация устройств визуализации и взаимодействия для иммерсивных сред. Устройства визуализации виртуальных объектов: VR шлемы, очки дополненной реальности, панели и мониторы для отображения виртуальных объектов. Устройства	**2 часа**

взаимодействия с виртуальными объектами в иммерсивных средах: системы трекинга головы, глаз, движений тела; перчатки, 3D контроллеры, устройства с обратной связью, платформы, датчики.	
Тема 3. Разработка приложений дополненной реальности Распознавание образов. Методы распознавания образов. Типы задач распознавания образов. Технологии дополненной реальности. Архитектура приложений дополненной реальности. Сферы применения дополненной реальности. Ограничения технологии дополненной реальности. Обзор средств разработки приложений дополненной реальности. Маркерные технологии дополненной реальности. Создание простейших статических и динамических QR-кодов.	**4 часа**
Тема 4. Разработка приложений виртуальной реальности Основы работы с SDK Unity 3D. Создание VR-приложения с использованием SDK Unity. Сенсоры, манипуляторы, устройства распознавания жестов. Программное обеспечения функционирования аппаратной составляющей взаимодействия с объектами виртуальной реальности. Использование Unity Web Player. Вопросы оптимизации.	**4 часа**
Тема 5. Разработка высокоэффективных приложений виртуальной и расширенной реальности Разница между AR, Virtual Reality (VR) и Mixed Reality. Оборудование. Ведущие компании-разработчики VR/AR проектов. Платформы для разработки приложений AR. Этапы разработки: выбор среды с учетом особенностей (мобильное приложение, промышленный или корпоративный контекст), выбор инструментальных средств, разработка дизайна, кодирование (отображение, взаимодействие, поддержка), тестирование. Технология разработки AR-приложения в Unity.	**4 часа**

II. СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

3 семестр

Лабораторные работы (36 часов)

Лабораторная работа №1. Основы Robot Operating System (ROS) Требуется установить ROS, собрать первый проект. Запустить turtle-bot и модуль управления с клавиатуры.	5 часов

Лабораторная работа №2. Основы Gazebo В качестве практического задания предполагается задачи в симуляторе Gazebo. Использовать любую существующую модель робота и среды. Сделать управление роботом с клавиатуры.	5 часов
Лабораторная работа №3. Алгоритмы планирования движением Реализовать алгоритм обхода всей доступной области в среде Gazebo и протестировать на любой модели робота.	5 часов
Лабораторная работа №4. SLAM Реализовать построение карты используя LIDAR. Реализовать построение карты используя карту глубин с камеры.	5 часов
Лабораторная работа №5. Летающие роботы UAV (БПЛА) В качестве практического задания предполагается задачи в симуляторе Gazebo, а также рассматривается вариант с выполнением на реальном роботе.	6 часов
Лабораторная работа №6. Обработка изображений, дополненная реальность (AR) В качестве практического задания рассматривается отслеживание камеры, отслеживание движущихся маркеров	5 часов
Лабораторная работа №7. Взаимодействие людей и роботов (HRI) Реализовать распознавание жестов	5 часов

4 семестр

Лабораторные работы (18 часов)

Лабораторная работа №1. Взаимодействие людей и роботов, AR. Требуется реализовать визуализацию решений, принимаемых роботом с помощью дополненной реальности.	6 часов
Лабораторная работа №2. Интерфейс управления роботом с помощью виртуальной реальности Реализовать визуализацию данных с камеры с помощью виртуальной реальности. Управление летательным аппаратом с помощью контроллеров HTC Vive или аналогичных.	6 часов
Лабораторная работа №3. Реализация на реальном роботе Реализация алгоритма обхода области не реальном	6 часов

летательном аппарате (дроне). Визуализация в VR.	

III. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Технологии виртуальной и дополненной реальности в промышленном производстве» представлено в Приложении 1 и включает в себя:

- план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;
- характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;
- требования к представлению и оформлению результатов самостоятельной работы;
- критерии оценки выполнения самостоятельной работы.

IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

№ п/п	Контролируемые разделы / темы дисциплины	Коды и этапы формирования компетенций		Оценочные средства	
				текущий контроль	промежуточная аттестация
1	Основы технологий виртуальной и дополненной реальности	ПК-8 ПК-13 ПК-16 ПК-19	знает умеет владеет	УО-1 ПР-11 (выполнение лабораторных работ)	Зачет с оценкой
2	Устройства визуализации и взаимодействия для иммерсивных сред	ПК-8 ПК-13 ПК-16 ПК-19	знает умеет владеет	УО-1 ПР-11 (выполнение лабораторных работ)	Зачет с оценкой
3	Разработка приложений дополненной реальности	ПК-8 ПК-13 ПК-16 ПК-19	знает умеет владеет	УО-1 ПР-11 (выполнение лабораторных работ)	Зачет с оценкой
4	Разработка приложений виртуальной реальности	ПК-8 ПК-13 ПК-16 ПК-19	знает	УО-1 ПР-11 (выполнение лабораторных работ)	Зачет с оценкой

5	Разработка высокоэффективных приложений виртуальной и расширенной реальности	ПК-8 ПК-13 ПК-16 ПК-19	знает	УО-1 ПР-11 (выполнение лабораторных работ)	Зачет с оценкой

1. устный опрос (УО): собеседование (УО-1), коллоквиум (УО-2); итоговая презентация (УО-3); круглый стол (УО-4);
2. технические средства контроля (ТС);
3. письменные работы (ПР): тесты (ПР-1), контрольные работы (ПР-2), эссе (ПР-3), рефераты (ПР-4), курсовые работы (ПР-5), научно-учебные отчеты по практикам (ПР-6), конспект (ПР-7), проект (ПР-9). Разноуровневые задачи и задания (ПР-11) и т.п.

Типовые контрольные задания, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература
(электронные и печатные издания)

1. Филлипс Ч., Харбор Р. Системы управления с обратной связью. – Лаборатория базовых знаний, 2001.

2. Carbone G., Gomez-Bravo F. Motion and operation planning of robotic systems //Springer International Publishing. Switzerland. DOI. – 2015. – Т. 10. – С. 978-3.

3. Джонатан, Л. Виртуальная реальность в Unity [Электронный ресурс] / Л. Джонатан ; пер. с англ. Р.Н. Рагимов. — Электрон. дан. — Москва : ДМК Пресс, 2016. — 316 с. — Режим доступа: https://e.lanbook.com/book/93271. — Загл. с экрана.

4. Иванцивская Н.Г. Перспектива. Теория и виртуальная реальность [Электронный ресурс]: учебное пособие/ Иванцивская Н.Г.— Электрон. текстовые данные.— Новосибирск: Новосибирский государственный

технический университет, 2010.— 197 с.— Режим доступа: http://www.iprbookshop.ru/44820.html.— ЭБС «IPRbooks»

5. Маров М. Н. 3ds max. Реальная анимация и виртуальная реальность/ Санкт-Петербург: Питер, 2006. – 414 с.

Дополнительная литература
(печатные и электронные издания)

1. Торн, А. Основы анимации в Unity [Электронный ресурс] / А. Торн ; пер. с англ. Р. Рагимова. — Электрон. дан. — Москва : ДМК Пресс, 2016. — 176 с. — Режим доступа: https://e.lanbook.com/book/73075. — Загл. с экрана.
2. Торн, А. Искусство создания сценариев в Unity [Электронный ресурс] : руководство / А. Торн ; пер. с англ. Р. Н. Рагимова. — Электрон. дан. — Москва : ДМК Пресс, 2016. — 360 с. — Режим доступа: https://e.lanbook.com/book/82812. — Загл. с экрана.
3. Дикинсон, К. Оптимизация игр в Unity 5 [Электронный ресурс] / К. Дикинсон. — Электрон. дан. — Москва : ДМК Пресс, 2017. — 306 с. — Режим доступа: https://e.lanbook.com/book/90109. — Загл. с экрана.
4. Вдовин А.С. Дизайн игр и медиаиндустрии. Персонажная графика и анимация [Электронный ресурс]: учебное пособие/ Вдовин А.С.— Электрон. текстовые данные.— Саратов: Саратовский государственный технический университет имени Ю.А. Гагарина, ЭБС АСВ, 2015.— 267 с.— Режим доступа: http://www.iprbookshop.ru/76480.html.— ЭБС «IPRbooks»
5. Смолин А.А., Жданов Д.Д., Потемин И.С., Меженин А.В., Богатырев В.А. Системы виртуальной, дополненной и смешанной реальности Учебное пособие. – Санкт- Петербург: Университет ИТМО. 2018 . – 59 с. https://books.ifmo.ru/file/pdf/2321.pdf
6. Фореман Н. ., Коралло Л. Прошлое и будущее 3D-технологий виртуальной реальности. Научно-технический вестник ИТМО. ноябрь-декабрь 2014. [Электронный ресурс]. Режим доступа http://ntv.ifmo.ru/ru/article/11182/proshloe_i_buduschee_3D_tehnologiy_virtualnoy_realnosti.htm
7. Виртуальная реальность. Единая коллекция цифровых образовательных ресурсов 2017 [Электронный ресурс]. Режим доступа http://files.schoolcollection.edu.ru/dlrstore/39131517-5991-11da-8314-0800200c9a66/index.htm

8. Полное погружение в виртуальную реальность: настоящее и будущее. 2017 [Электронный ресурс]. Режим доступа https://habrahabr.ru/company/miip/blog/330754/
9. Виртуальная реальность (VR): прошлое, настоящее и будущее 2017 [Электронный ресурс]. Режим доступа http://vrmania.ru/stati/virtualnaya-realnost.html
10. 12 платформ разработки приложений дополненной реальности 2017 [Электронный ресурс]. Режим доступа https://apptractor.ru/info/articles/12-platform-razrabotki-prilozheniy-dopolnennoyrealnosti.html

Перечень ресурсов информационно-телекоммуникационной сети «Интернет»

1. Robot Operating System (ROS) — это гибкая среда для написания программного обеспечения для роботов. Это набор инструментов, библиотек и соглашений, призванных упростить задачу создания сложного и надежного поведения робота на самых разных роботизированных платформах: www.ros.org
2. http://gazebosim.org/
3. www.generationrobots.com/en/content/83-carry-out-simulations-and-make-your-darwin-op-walk-with-gazebo-and-ros
4. https://robotacademy.net.au/ – Открытые лекции от автора курса «Введение в робототехнику», Квинслендский технологический университет.

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

VII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Компьютерный класс: Проектор DLP, 3000 ANSI Lm, WXGA 1280x800, 2000:1 EW330U Mitsubishi; Системный блок с монитором. Процессор: Intel I5-8600k 3.6Ghz, оперативная память: 32gb, жесткий диск: 1ТБ, графический ускоритель: Nvidia GTX 1080 Беспроводные ЛВС для обучающихся обеспечены системой на базе точек доступа 802.11a/b/g/n 2x2 MIMO(2SS). **Специализированное ПО**: Visual Studio 2019, Unity 2019, STEAM VR, Anaconda, Oculus Rift SDK, HTC VIVE SDK	690922, Приморский край, г. Владивосток, о. Русский, п. Аякс, 10, г. Владивосток, о. Русский, п. Аякс, корпус G, ауд. G464

Приложение 1

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

по дисциплине

ТЕХНОЛОГИИ ВИРТУАЛЬНОЙ И ДОПОЛНЕННОЙ РЕАЛЬНОСТИ
В ПРОМЫШЛЕННОМ ПРОИЗВОДСТВЕ

Направление подготовки

09.04.01 Информатика и вычислительная техника

магистерская программа
«Технологии виртуальной и дополненной реальности»

Форма подготовки очная

Владивосток
2018

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Технологии виртуальной и дополненной реальности в промышленном производстве» включает в себя:

- план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;
- характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;
- требования к представлению и оформлению результатов самостоятельной работы;
- критерии оценки выполнения самостоятельной работы.

План-график выполнения самостоятельной работы по дисциплине

№ п/п	Дата/сроки выполнения	Вид самостоятельной работы	Примерные нормы времени на выполнение	Форма контроля
	1 семестр		**72 часа**	
1	1 – 17 недели	Подготовка к лабораторным работам	66 часов	Выполнение лабораторных работ
2	18 неделя	Подготовка к зачету	6 часов	Зачет с оценкой
	2 семестр		**72 часа**	
1	1 – 17 недели	Подготовка к лабораторным работам	30 часов	Выполнение лабораторных работ
2	18 неделя	Подготовка к зачету	6 часов	Зачет с оценкой

Примеры заданий для самостоятельной работы

Игровые объекты и скрипты

Создание часов

- *Соберите часы с простыми предметами.*
- *Напишите скрипт на C #.*
- *Вращайте стрелки часов, чтобы показать время.*
- *Animate the arms..*

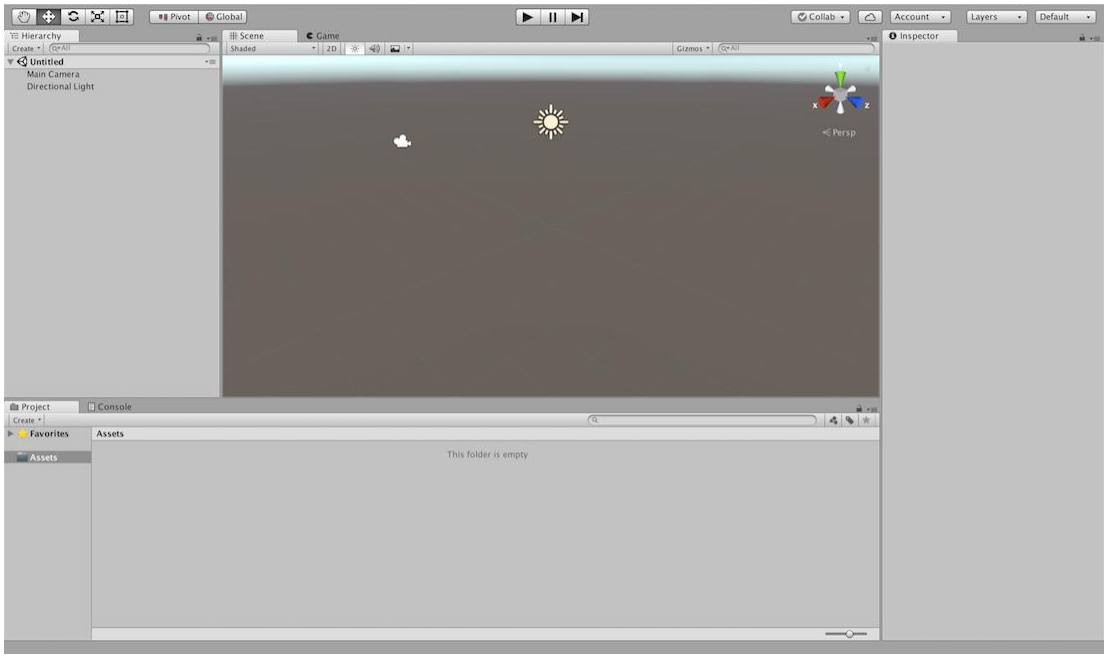
В этом уроке мы создадим простые часы и запрограммируем компонент, чтобы он отображал текущее время. Вам нужно только минимальное понимание редактора Unity. Если знакомы с интерфейсом и знаете, как перемещаться по окну *сцены,* тогда все готово.

В этом руководстве предполагается, что вы используете хотя бы Unity 2017.1.0.

Настало время создать часы.

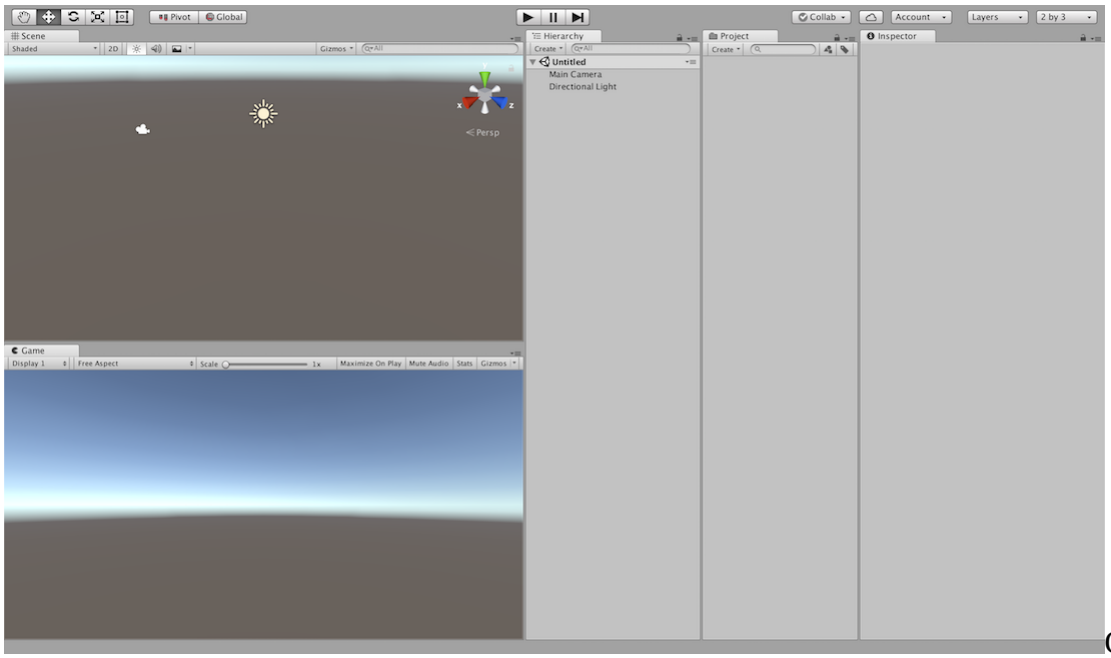
Строим простые часы

Откройте Unity и создайте новый 3D-проект. Вам не нужны никакие дополнительные пакеты активов, и вам также не нужна аналитика. Если вы еще не настроили редактор, вы получите его стандартное расположение окон.



Макет окна по умолчанию.

I use a different layout, the *2 by 3* preset which you can select from the dropdown list at the top right corner of the editor. I customize that one further by switching the *Project* window to *One Column Layout*, which better fits its vertical orientation. You can change it via the small dropdown list next to it's lock icon, at the top right of the window above its toolbar. I also disabled *Show Grid* in the *Scene* window, via its *Gizmos* dropdown menu.



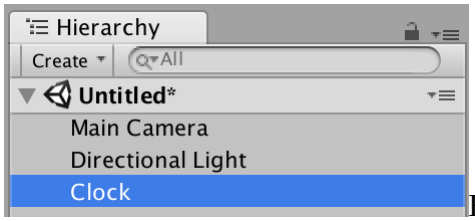
Customized 2-by-3 layout.

Why is my *Game* window small with a dark border?

Creating a Game Object

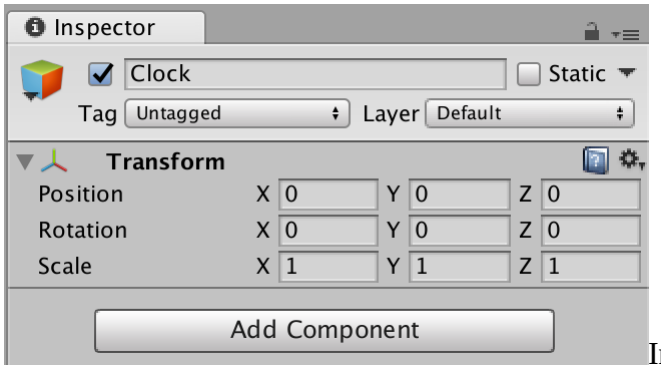
The default scene contains two game objects. They are listed in the *Hierarchy* window and you can also see their icons in the *Scene* window. First is the *Main Camera*, which is used to render the scene. The *Game* window is rendered using this camera. Second is the *Directional Light*, which illuminates the scene.

To create your own game object, use the *GameObject / Create Empty* option. You can also do this via the context menu of the *Hierachy* window. This adds a new object to the scene, which you can immediately give a name. As we're going to create a clock, name it *Clock*.



Hierarchy with clock object.

The *Inspector* window shows the details of game objects. When our clock object is selected, it will contain a header with the object's name plus a few configuration options. By default, the object is enabled, is not static, doesn't have a tag, and belongs to the default layer. These settings are fine for us. Below that, it shows a list of all the components of the game object. There is always a *Transform*component, which is all our clock currently has.



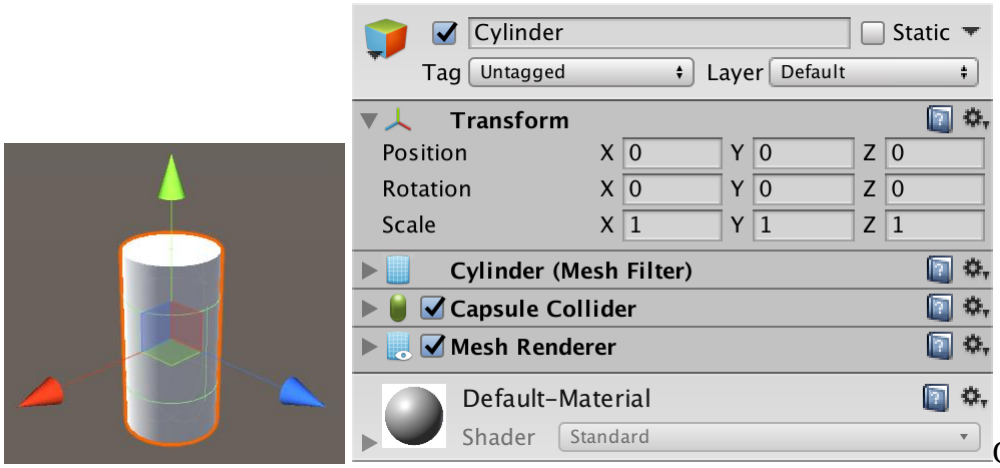
Inspector window with clock selected.

The *Transform* component contains the position, rotation, and scale of the object in 3D space. Make sure that the clock's position and rotation are both 0. Its scale should be 1.

What about 2D objects?

Creating the Face of the Clock

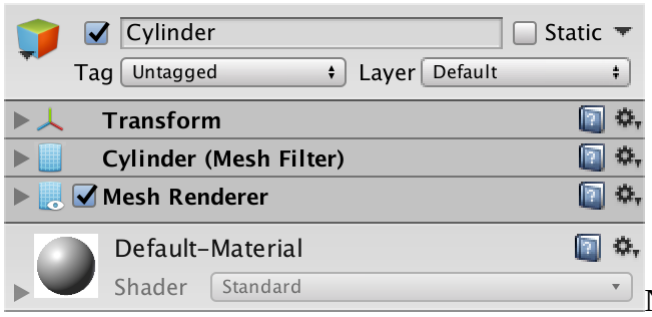
Although we have a clock object, we don't see anything yet. We'll have to add 3D models to it so they get rendered. Unity contains a few primitive objects that we can use to build a simple clock. Let's begin by adding a cylinder to the scene via *GameObject / 3D Object / Cylinder*. Make sure that it has the same *Transform* values as our clock.



Game object representing a cylinder.

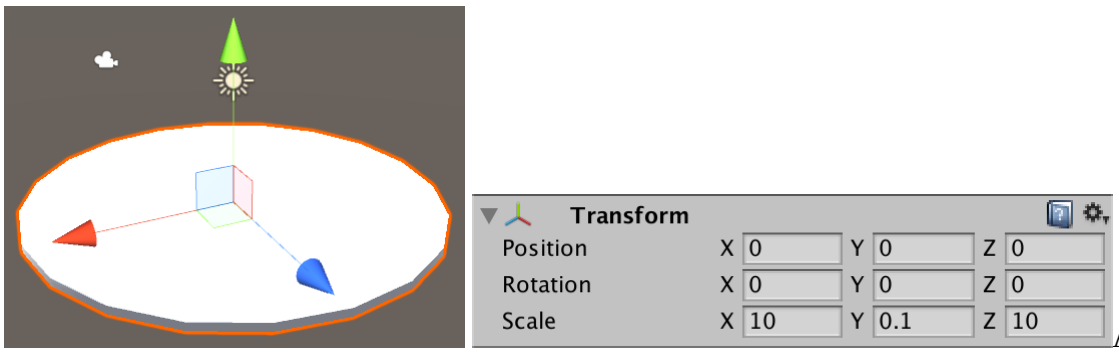
The new object has three more components than an empty game object. First, it has a *Mesh Filter*, which simply contains a reference to the built-in cylinder mesh. Second is a *Capsule Collider*, which is for 3D physics. Third is a *Mesh Renderer*. That component is there to ensure that the object's mesh gets rendered. It also controls which material is used for rendering, which is the built-in *Default-Material*unless you change it. This material is also shown in the inspector, below the component list.

Although the object represents a cylinder, it has a capsule collider, because Unity doesn't have a primitive cylinder collider. We don't need it, so we can remove that component. If you'd like to use physics with your clock, you're better off using a *Mesh Collider* component. Components can be removed via the dropdown menu with a gear icon, in their top right corner.



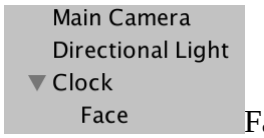
No more collider.

To turn the cylinder into a clock face, we have to flatten it. This is done by decreasing the Y component of its scale. Reduce it to 0.1. As the cylinder mesh is two units high, its effective height becomes 0.2 units. Let's also make a big clock, so increase the X and Z components of its scale to 10.



A scaled cylinder.

Change the name of the cylinder object to *Face*, as it represents the face of the clock. It is part of the clock, so make it a child of the *Clock* object. You can do this by dragging the face onto the clock in the *Hierarchy* window.



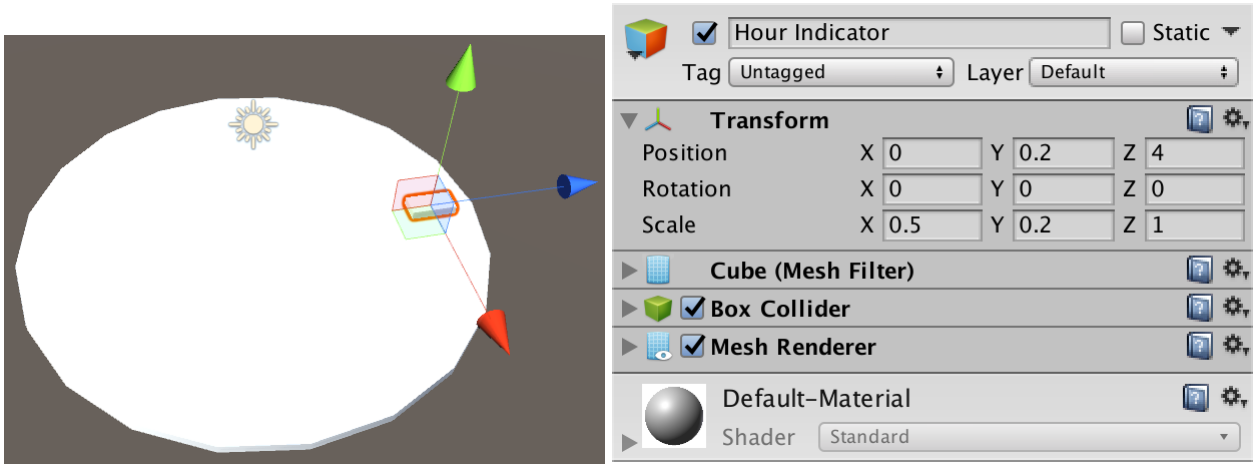
Face child object.

Child objects are subject to the transformation of their parent object. This means that when *Clock*changes position, *Face* does as well. It's as if they were a single entity. The same goes for rotation and scale. You can use this to make complex object hierarchies.

Creating the Clock Periphery

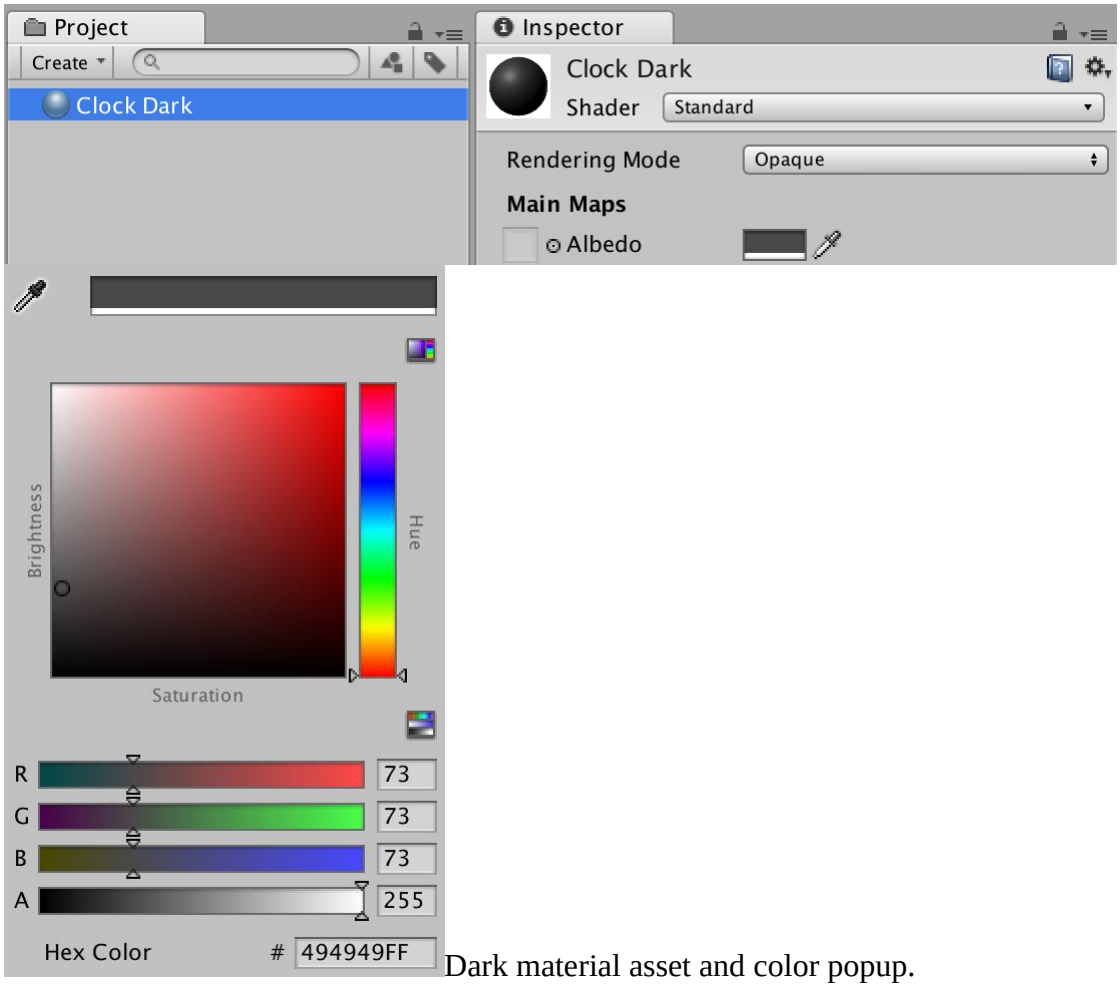
The outer ring of a clock's face usually has markings that help indicate what time it is displaying. This is known as the clock periphery. Let's use blocks to indicate the hours of a 12-hour clock.

Add a cube object to the scene via *GameObject / 3D Object / Cube*. Change its scale to (0.5, 0.2, 1) so it becomes a narrow flat long block. It is now located inside the clock's face. Set its position to (0, 0.2, 4). That places it on top of the face and to the side that corresponds with the 12th hour. Name it *Hour Indicator*.



Indicator for the 12th hour.

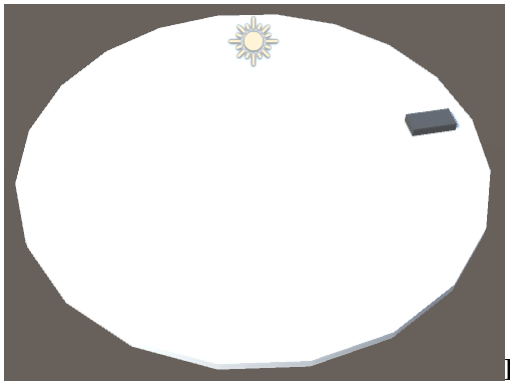
The indicator is hard to see, because it has the same color as the face. Let's create a separate material for it, via *Assets / Create / Material*, or via the context menu of the *Project* window. This gives us a material asset that is a duplicate of the default material. Change its *Albedo* to something darker, like 73 for red, green, and blue. That results in a dark gray material. Give it an appropriate name, like *Clock Dark*.



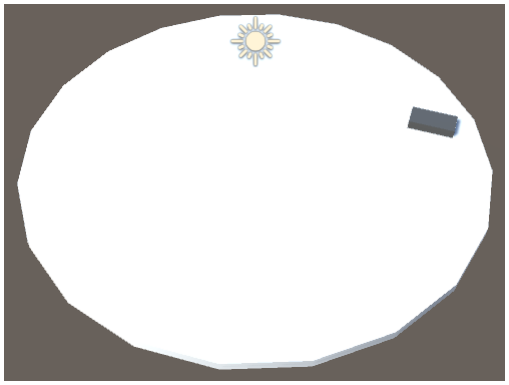
Dark material asset and color popup.

What is albedo?

Make the hour indicator use this material. You can do this by dragging the material onto the object in either the scene or hierarchy window. You can also drag it to the bottom of the inspector window, or change *Element 0* of the mesh renderer's *Materials* array.


Dark hour indicator.

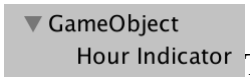
Our indicator is correctly positioned for hour 12, but what if we wanted to indicate hour 1? As there are twelve hours and a full circle has 360°, we'd have to rotate the indicator 30° around the Y axis. Let's give that a try.



Rotated hour indicator, incorrectly positioned.

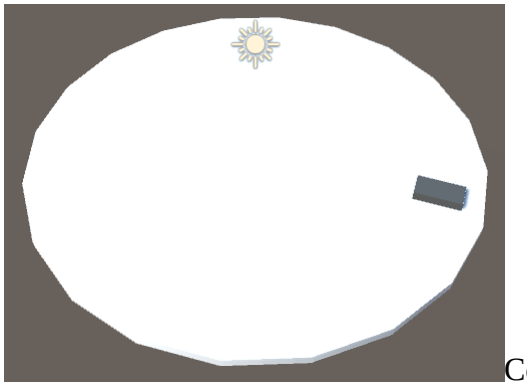
While this gives us the correct orientation, the indicator is still in the position for hour 12. This is the case because an object's rotation is relative to its own local origin, which is its position.

We have to move the indicator along the edge of the face to align it with hour 1. Instead of figuring out this position ourselves, we can use the object hierarchy to do this for us. First reset the indicator's rotation to 0. Then create a new empty object, with position and rotation 0 and scale 1. Make the indicator a child of that object.

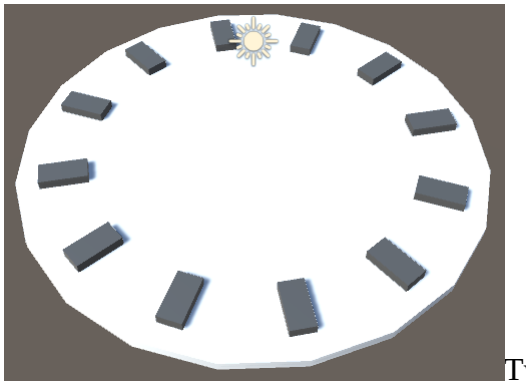


Temporary parent.

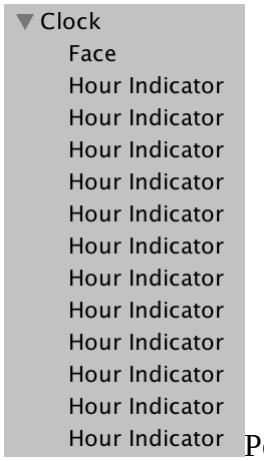
Now set the parent's Y rotation to 30°. The indicator will rotate as well, effectively orbiting it's parent's origin, and ends up exactly where we want it to be.


Correctly positioned hour indicator.

Duplicate the temporary parent, with Control or Command D, or via the context menu in the hierarchy. Increase the Y rotation of the duplicate by another 30°. Keep doing this until you end up with one indicator per hour.


Twelve hour indicators.

We no longer need the temporary parents. Select one of the hour indicators in the hierarchy and drag it onto the clock. It has now become a child of the clock. When this happened, Unity changed the indicator's transformation so its position and rotation didn't change in world space. Repeat this for all twelve indicators, then delete the temporary parent objects. You can speed this up by selecting multiple objects at the same time, via control- or command-clicking.

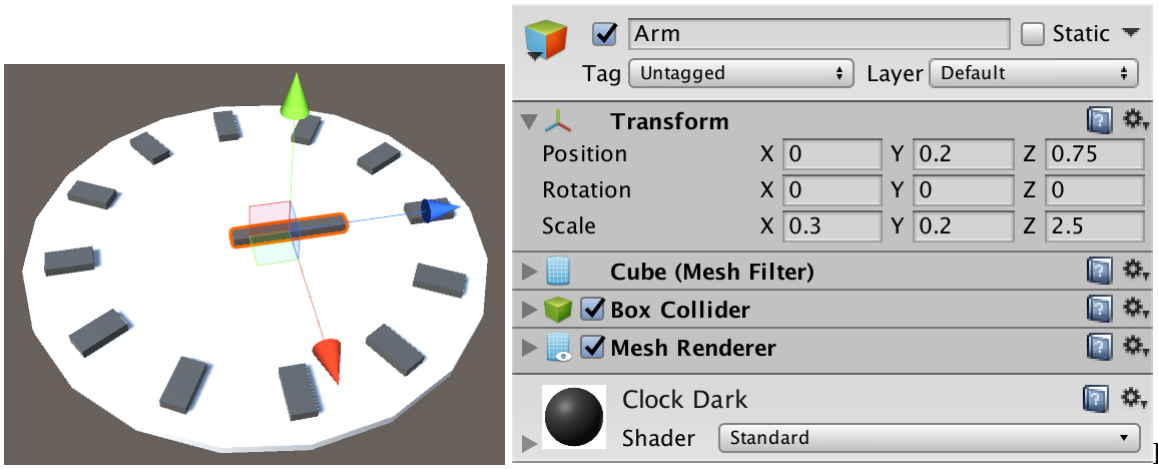


Periphery children.

I'm seeing values like 90.00001. Is that a problem?

Creating the Arms

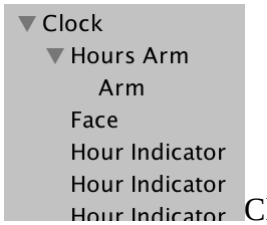
We can use the same approach to construct the arms of the clock. Create another cube named *Arm*and give it the same dark material that the indicators use. Set its scale to (0.3, 0.2, 2.5) so it's narrower and longer than the indicators. Set its position to (0, 0.2, 0.75) so it sits on top of the face and points towards hour 12, but also a bit in the opposite direction. That makes it look as if the arm has a little counterweight when it rotates.



Hours arm.

Where did the light's icon go?

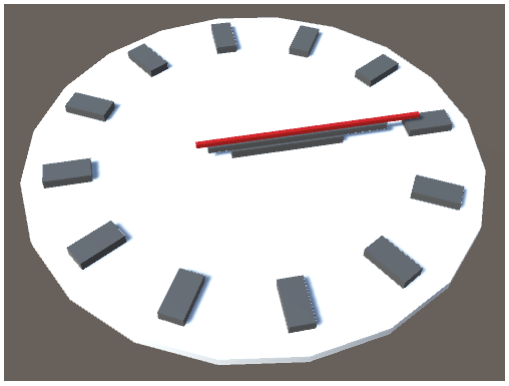
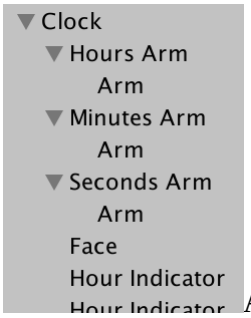
To make the arm pivot around the center of the clock, create a parent object for it, like we did to position the hour indicators. Again make sure that its transformation has the default values of 0 for position and rotation and 1 for scale. Because we'll be rotating the arm later, make this parent a child of the clock and name it *Hours Arm*. So *Arm* ends up as a grandchild of *Clock*.



Clock hierarchy with hours arm.

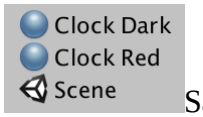
Duplicate *Hours Arm* twice to create *Minutes Arm* and *Seconds Arm*. The minutes arm should be narrow and longer than the hours arm, so set its *Arm* child object's scale to (0.2, 0.15, 4) and its position to (0, 0.375, 1). That way it ends up on top of the hours arm.

For the child of *Seconds Arms*, use scale (0.1, 0.1, 5) and position (0, 0.5, 1.25). To differentiate it further, I created a *Clock Red* material with the RGB values of its albedo set to (197, 0, 0) and made the *Arm* child use that.

All three arms.

Our clock has now been constructed. If you haven't done so already, this is a good moment to save the scene. It will be stored as an asset in the project.



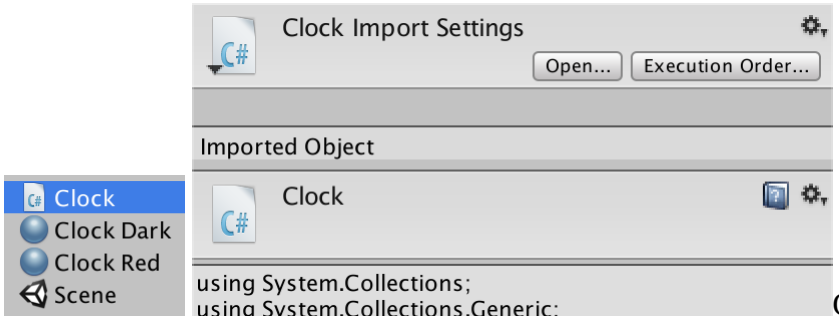
Saved scene.

If you are stuck, want to compare, or would like to skip building the clock, you can download a packagecontaining the work done is this section. You can import these packages into a Unity project via *Assets / Import Package / Custom Package...*, dragging it onto the Unity window, or double-clicking it in your file browser.

unitypackage

Animating the Clock

Our clock currently does not tell the time. It's just an object hierarchy, which causes Unity to render a bunch of meshes. Nothing more. Had there been a default clock component, we could've used that to do some time-telling. As there isn't one, we'll have to create our own. Components are defined via scripts. Add a new script asset to the project via *Assets / Create / C# Script* and name it *Clock*.



Clock script asset.

When the script is selected, the inspector will show its contents, and a button to open the file in a code editor. You can also double-click the asset to open the editor. The script file will contain the default code template for a component, shown below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Clock : MonoBehaviour {

        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
}
```

This is C# code. It's the programming language used for scripting in Unity. To understand how the code works, we'll delete it all and start from scratch.

What about JavaScript?

Defining a Component Type

An empty file is not a valid script. It must contain the definition of our clock component. We don't define a single instance of a component. Instead, we define the general class or type known as `Clock`. Once that's established, we could create multiple such components in Unity.

In C#, we define the `Clock` type by first stating that we're defining a class, followed by its name. In the code fragments below, changed code has a yellow background. As we start with an empty file, the contents of it should literally become `class Clock` and nothing else, though you could add spaces and newlines between words as you like.

```
class Clock
```

What's a class, technically?

Because we don't want to restrict which code has access to our `Clock`, it is good form to prefix it with the `public` access modifier.

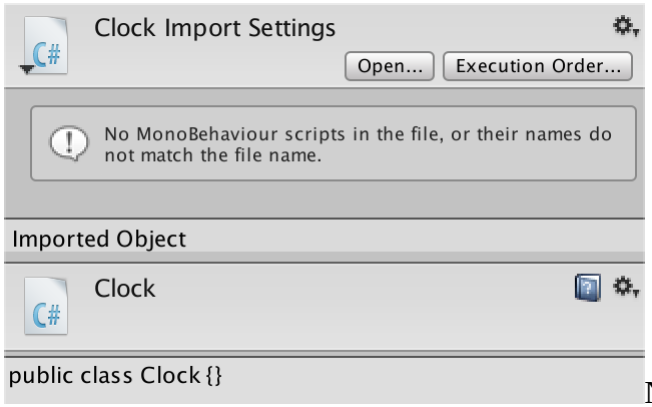
```
public class Clock
```

What is the default access modifier for classes?

At this point we don't have valid C# syntax yet. We indicated that we're defining a type, so we must actually define what it is like. That's done by a block of code that follows the declaration. The boundaries of a code block are indicated with curly brackets. We're leaving it empty for now, so just use `{}`.

```
public class Clock {}
```

Our code is now valid. Save the file and switch back to Unity. The Unity editor will detect that the script asset has changed and triggers a recompilation. After that is done, select our script. The inspector will inform us that the asset does not contain a **`MonoBehaviour`** script.



Non-component script.

What this means is that we cannot use this script to create a component in Unity. At this point, our **`Clock`**defines a generic C# object type. Unity can only use subtypes of **`MonoBehaviour`** to create components.

What does mono-behavior mean?

To turn **`Clock`** into a subtype of **`MonoBehaviour`**, we have to change our type declaration so that it extends that type, which is done with a colon. This makes **`Clock`** inherit all the functionality of **`MonoBehaviour`**.

```
public class Clock : MonoBehaviour {}
```

However, this will result in an error after compilation. The compiler complains that it cannot find the **`MonoBehaviour`** type. This happens because the type is contained in a namespace, which is `UnityEngine`. To access it, we have to use its fully-qualified name, `UnityEngine.`**`MonoBehaviour`**.

```
public class Clock : UnityEngine.MonoBehaviour {}
```

What's a namespace?

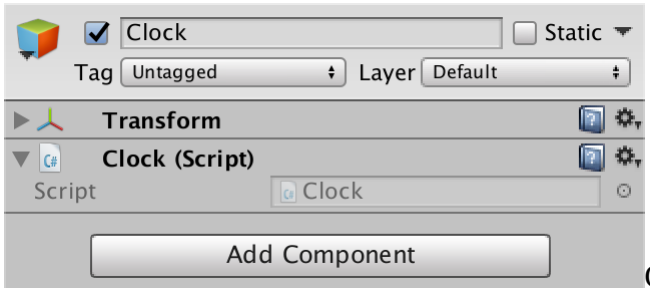
Because it is inconvenient to always have to use the `UnityEngine` prefix when accessing Unity types, we can tell the compiler to search this namespace when we don't explicitly mention any.

This is done by adding `using UnityEngine;` at the top of the file. The semicolon is required to mark the end of the command.

```
using UnityEngine;

public class Clock : MonoBehaviour {}
```

Now we can add our component to our clock game object in Unity. This can be done either by dragging the script asset onto the object, or via the *Add Component* button at the bottom of the object's inspector.



Clock with our component.

A C# object instance has now been created, using our **`Clock`** class as a template. It has been added to the component list of the *Clock* game object.

Getting Hold of an Arm

To rotate the arms, **`Clock`** objects need to know about them. Let's start with the hours arm. Like all game objects, it can be rotated by changing the rotation of its transform component. So we have to add knowledge of the hour arm's transform component to **`Clock`**. This can be done by adding a data field inside its code block, defined as a name followed by a semicolon.

Hours transform would be an appropriate name. However, names have to be single words. The convention is to make the first word of a field name lowercase and capitalize all other words, then stick them together. So it becomes `hoursTransform`.

```
public class Clock : MonoBehaviour {

        hoursTransform;
}
```

Where did the `using` statement go?

We also have to define the type of the field, which in this case is `UnityEngine.`**`Transform`**. It has to be placed in front of the name's field.

```
        Transform hoursTransform;
```

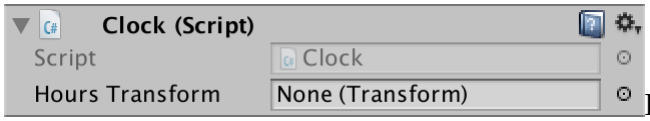
Our class now defines a field that can hold a reference to another object, whose type has to be **`Transform`**. We have to make sure that it holds a reference to the transform component of the hours arm.

Fields are private by default, which means that they can only be accessed by the code belonging to `Clock`. But the class doesn't know about our Unity scene. By making the field public, anyone can change its contents.

```
	public Transform hoursTransform;
```

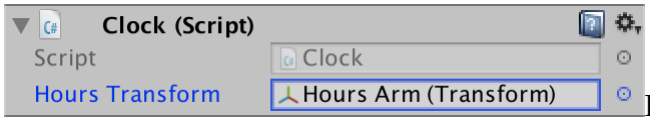
Aren't public fields bad form?

Once the field is public, it will show up in the inspector window. This happens because the inspector automatically makes all public fields of components editable.



Hours transform field.

To make the proper connection, drag the *Hours Arm* from the hierarchy onto the *Hours Transform* field. Alternatively, use the circular button at the right of the field and search for *Hours Arm*.



Hours transform connected.

After dragging or selecting the hours arm object, the Unity editor grabs its transform component and puts a reference to it in our field.

Knowing all Three Arms

We have to do the same for the minutes and seconds arms. So add two more appropriately-named fields to `Clock`.

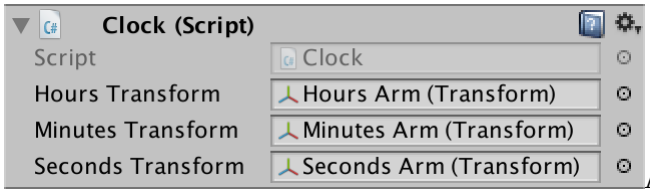
```
	public Transform hoursTransform;
	public Transform minutesTransform;
	public Transform secondsTransform;
```

It is possible to make these field declarations more concise, because they share the same access modifier and type. They can be consolidated into a comma-separated list of field names following the access modifier and type.

```
	public Transform hoursTransform, minutesTransform, secondsTransform;
//	public Transform minutesTransform;
//	public Transform secondsTransform;
```

What does `//` do?

Hook up the other two arms in the editor as well.



All three arms connected.

Knowing the Time

Now that we can reach the arms in **`Clock`**, the next step is to figure out the current time. To do this, we need to tell **`Clock`** to execute some code. This is done by adding a code block to the class, known as a method. The block has to be prefixed by a name, which is capitalized by convention. We'll name it **`Awake`**, suggesting that the code should be executed when the component awakens.

```
public class Clock : MonoBehaviour {

	public Transform hoursTransform, minutesTransform, secondsTransform;

	Awake () {
	}
}
```

Methods are somewhat like mathematical functions, for example $f(x)=2x+3f(x)=2x+3$. That function takes a number, doubles it, then adds three. It operates on a single number, and its result is a single number as well. In the case of a method, it's more like $f(p)=cf(p)=c$ where pp represents input parameters and ccrepresents code execution. As that's rather generic, what is the result of such a function? That has to be mentioned explicitly. In our case, we just want to execute some code, without providing a resulting value. In other words, the result of the method is void. We indicated this with the **`void`** prefix.

```
	void Awake () {}
```

We also don't need any input data. However, we still have to define the method's parameters, as a comma-separated list between round brackets. It's just empty in our case.

```
	void Awake () {}
```

We now have a valid method, although it doesn't do anything yet. Like Unity detected our fields, it also detects this **`Awake`** method. When a component has an **`Awake`** method, Unity will invoke that method when the component awakens. This happens after it's been created or loaded.

Doesn't `Awake` have to be public?

To test whether this works, let's have **`Awake`** create a debug message.
The `UnityEngine.Debug` class contains a publicly-available `Log` method for this purpose. We'll pass it a simple string of text to print. Strings are written between double quotes. Again, a semicolon is required to mark the end of an expression.

```
	void Awake () {
		Debug.Log("Test");
	}
```

Enter play mode in the Unity editor. You'll see the test string show up in the status bar at the bottom of the editor. You can also see it in the console window, which you can open via *Window / Console*. The console provides some additional information, like which code generated the message, when you select the logged text.

Now that we know that our method works, let's figure out the current time when it is invoked. The `UnityEngine` namespace contains a **`Time`** class, which in turn contains a `time` property. It seems obvious to use that, so let's log it.

```
	void Awake () {
		Debug.Log(Time.time);
	}
```

What's a property?

It turns out to always log the value 0. That's because **`Time`**`.time` tells us how many seconds have passed since entering play mode. As our **`Awake`** method gets invoked immediately, no time has passed yet. So this doesn't tell us the real time.

To access the system time of the computer we're running on, we can use the **`DateTime`** structure. This isn't a Unity type, it is found in the `System` namespace. It's part of the core functionality of the .NET framework, which is what Unity uses to support scripting.

What is a structure?

`DateTime` has a publicly-accessible `Now` property. It produces a **`DateTime`** value that contains the current system date and time. Let's log it.

```
using System;
using UnityEngine;

public class Clock : MonoBehaviour {

	public Transform hoursTransform, minutesTransform, secondsTransform;

	void Awake () {
		Debug.Log(DateTime.Now);
	}
}
```

Now we get a timestamp logged each time we enter play mode.

Rotating the Arms

The next step is to rotate the clock's arms based on the current time. Let's again start with the hours. **`DateTime`** has an `Hour` property that gets us the hours portion of a **`DateTime`** value. Invoking it on the current timestamp will give us the hour of the day.

```
	void Awake () {
		Debug.Log(DateTime.Now.Hour);
	}
```

We can use that to construct a rotation. Rotations are stored in Unity as quaternions. We can create one via the publicly-available **`Quaternion`**`.Euler` method. It has regular angles for the X, Y, and Z axis as parameters and produces an appropriate quaternion.

```
	void Awake () {
```

```
//              Debug.Log(DateTime.Now.Hour);
                Quaternion.Euler(0, DateTime.Now.Hour, 0);
        }
```

What's a quaternion?

All three parameters are real numbers, which are represented in C# by floating-point values. To explicitly declare that we're supplying the method with such numbers, let's add the f suffix to both zeros.

```
                Quaternion.Euler(0f, DateTime.Now.Hour, 0f);
```

Our clock has twelve hour indicators set at 30° intervals. To make the rotation match that, we have to multiply the hours by 30. Multiplication is done with an asterisk.

```
                Quaternion.Euler(0f, DateTime.Now.Hour * 30f, 0f);
```

To make it clear that we're converting from hours to degrees, we can define an appropriately-named field for the conversion factor. As it is a floating-point value, it's type is **`float`**. Because we already know the number, we can immediately assign it as part of the field declaration.

```
        float degreesPerHour = 30f;

        public Transform hoursTransform, minutesTransform, secondsTransform;

        void Awake () {
                Quaternion.Euler(0f, DateTime.Now.Hour * degreesPerHour, 0f);
        }
```

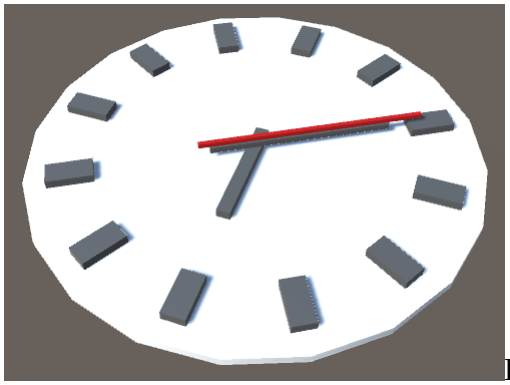
The amount of degrees per hour isn't supposed to change. We can enforce this by adding the **`const`**prefix to our declaration. This turns `degreesPerHour` into a constant.

```
        const float degreesPerHour = 30f;
```

What's special about constants?

At this point we have a rotation, but don't do anything with it yet, so it's simply discarded. To apply it to the hours arm, we have to assign it to the `localRotation` property of it's transform component.

```
        void Awake () {
                hoursTransform.localRotation =
                        Quaternion.Euler(0f, DateTime.Now.Hour * degreesPerHour,
0f);
        }
```


Hours arm at 4 o'clock.

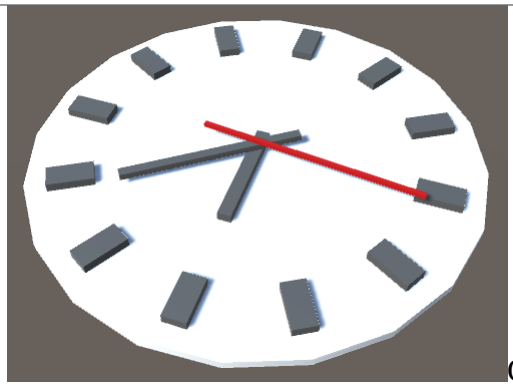
What's local about the rotation?

The hours arm now jumps to the correct orientation when entering play mode. Let's give the same treatment to the other two arms. Both a minute and a second take up six degrees.

```
	const float
		degreesPerHour = 30f,
		degreesPerMinute = 6f,
		degreesPerSecond = 6f;

	public Transform hoursTransform, minutesTransform, secondsTransform;

	void Awake () {
		hoursTransform.localRotation =
			Quaternion.Euler(0f, DateTime.Now.Hour * degreesPerHour,
0f);
		minutesTransform.localRotation =
			Quaternion.Euler(0f, DateTime.Now.Minute *
degreesPerMinute, 0f);
		secondsTransform.localRotation =
			Quaternion.Euler(0f, DateTime.Now.Second *
degreesPerSecond, 0f);
	}
```


Clock displaying 16:29:06.

We're using `DateTime.Now` three times, to retrieve the hour, minute, and second. Each time we go through the property again, which requires some work, which could theoretically result in different time values. To make sure that this doesn't happen, we should retrieve the time only once. We can do this by declaring a variable inside the method and assign the time to it, then use this value afterwards.

What's a variable?

```
	void Awake () {
		DateTime time = DateTime.Now;
```

```
		hoursTransform.localRotation =
			Quaternion.Euler(0f, time.Hour * degreesPerHour, 0f);
		minutesTransform.localRotation =
			Quaternion.Euler(0f, time.Minute * degreesPerMinute,
0f);
		secondsTransform.localRotation =
			Quaternion.Euler(0f, time.Second * degreesPerSecond,
0f);
	}
```

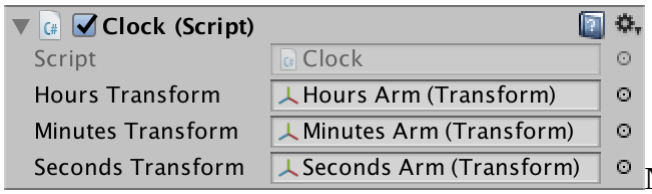
Animating the Arms

We get the current time when entering play mode, but after that the clock remains motionless. To keep the clock synchronized with the current time, change the name of our **`Awake`** method to **`Update`**. This method gets invoked by Unity every frame instead of just once, as long as we stay in play mode.

```
	void Update () {
		DateTime time = DateTime.Now;
		hoursTransform.localRotation =
			Quaternion.Euler(0f, time.Hour * degreesPerHour, 0f);
		minutesTransform.localRotation =
			Quaternion.Euler(0f, time.Minute * degreesPerMinute,
0f);
		secondsTransform.localRotation =
			Quaternion.Euler(0f, time.Second * degreesPerSecond,
0f);
	}
```

Clock that stays up to date.

Note that our component has also gained a toggle in front of its name in the inspector. This allows us to disable the component, which prevents Unity from invoking its **`Update`** method.



Now with enabled toggle.

Continuously Rotating

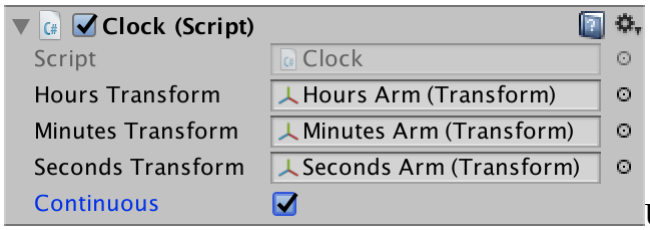
The arms of our clock indicate exactly the current hour, minute, or second. It behaves like a digital clock, discrete but with arms. Many clocks have slowly-rotating arms that provide an analog representation of time. Either approach works, so let's make this configurable by adding a toggle to our component's inspector.

Add another public field to **`Clock`**, named `continuous`. It can be either on or off, for which we can use the boolean type, declared with **`bool`**.

```
	public Transform hoursTransform, minutesTransform, secondsTransform;

	public bool continuous;
```

Booleans values are either `true` or `false`, which correspond to on and off in our case. They're false by default, so turn it on once the field appears in the inspector.



Using the continuous option.

We now have two approaches to support. To prepare for this, duplicate our `Update` method and rename them to `UpdateContinuous` and `UpdateDiscrete`.

```
	void UpdateContinuous () {}
		DateTime time = DateTime.Now;
		hoursTransform.localRotation =
			Quaternion.Euler(0f, time.Hour * degreesPerHour, 0f);
		minutesTransform.localRotation =
			Quaternion.Euler(0f, time.Minute * degreesPerMinute,
0f);
		secondsTransform.localRotation =
			Quaternion.Euler(0f, time.Second * degreesPerSecond,
0f);
	}

	void UpdateDiscrete () {
		DateTime time = DateTime.Now;
		hoursTransform.localRotation =
			Quaternion.Euler(0f, time.Hour * degreesPerHour, 0f);
		minutesTransform.localRotation =
			Quaternion.Euler(0f, time.Minute * degreesPerMinute,
0f);
		secondsTransform.localRotation =
			Quaternion.Euler(0f, time.Second * degreesPerSecond,
0f);
	}
```

Create a new `Update` method. If `continuous` is true, then it should invoke `UpdateContinuous`. This can be done with an `if` statement. The `if` keyword is followed by an expression within round brackets. If that expression evaluates as true, then the code block following it is executed. Otherwise, the code block is skipped.

```
	void Update () {
		if (continuous) {
			UpdateContinuous();
		}
	}
```

Where does the new `Update` method have to be defined?

It is also possible to add an alternative code block, to be executed when the expression ends up false. This is done with the `else` keyword. We can use that to invoke our `UpdateDiscrete` method.

```
	void Update () {
		if (continuous) {
```

```
                        UpdateContinuous();
                }
                else {
                        UpdateDiscrete();
                }
        }
```

We can now switch between approaches, but both still do the same thing. We have to adjust `UpdateContinuous` so it displays fractional hours, minutes, and seconds. Unfortunately, **`DateTime`** doesn't contain convenient fractional data. Fortunately, it does have a `TimeOfDay` property. This gives us a **`TimeSpan`** value that contains the data in the format that we need. Specifically `TotalHours`, `TotalMinutes`, and `TotalSeconds`.

```
        void UpdateContinuous () {
                TimeSpan time = DateTime.Now.TimeOfDay;
                hoursTransform.localRotation =
                        Quaternion.Euler(0f, time.TotalHours * degreesPerHour,
0f);
                minutesTransform.localRotation =
                        Quaternion.Euler(0f, time.TotalMinutes *
degreesPerMinute, 0f);
                secondsTransform.localRotation =
                        Quaternion.Euler(0f, time.TotalSeconds *
degreesPerSecond, 0f);
        }
```

This will result in compile errors, because the new values have the wrong type. They are defined as double-precision floating point values, known as **`double`**. These values provide higher precision than **`float`** values, but Unity's code only works with single-precision floating point values.

Is single precision enough?

We can solve this problem by converting from **`double`** to **`float`**. This simply discards the precision data that we do not need. This process is known as casting and is done by writing the new type within round brackets in front of the value to be converted.

```
                hoursTransform.localRotation =
                        Quaternion.Euler(0f, (float)time.TotalHours *
degreesPerHour, 0f);
                minutesTransform.localRotation =
                        Quaternion.Euler(0f, (float)time.TotalMinutes *
degreesPerMinute, 0f);
                secondsTransform.localRotation =
                        Quaternion.Euler(0f, (float)time.TotalSeconds *
degreesPerSecond, 0f);
```

Часы с непрерывно вращающимися стрелками.

Теперь вы знаете основы создания объектов и сценариев в Unity. Следующим шагом является построение графика .

Приложение 2

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЦИФРОВОЙ ЭКОНОМИКИ

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине

ТЕХНОЛОГИИ ВИРТУАЛЬНОЙ И ДОПОЛНЕННОЙ РЕАЛЬНОСТИ
В ПРОМЫШЛЕННОМ ПРОИЗВОДСТВЕ

Направление подготовки

09.04.01 Информатика и вычислительная техника

магистерская программа
«Технологии виртуальной и дополненной реальности»

Форма подготовки очная

Владивосток
2018

Фонд оценочных средств по дисциплине «Технологии виртуальной и дополненной реальности в промышленном производстве» включает в себя:

- *типовые контрольные задания,*
- *методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности,*
- *а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы*

Критерии оценивания результатов контрольно-оценочных мероприятий текущей и промежуточной аттестации по дисциплине

Система критериев оценивания, как и при проведении промежуточной аттестации по модулю, опирается на три уровня освоения компонентов компетенций: пороговый, повышенный, высокий.

Результаты обучения (компетенции из ФГОС)	Знает	Умеет	Владеет
ПК - 8 ПК – 13 ПК – 16 ПК – 19	технологии и алгоритмы виртуальной (VR) и дополненной реальности (AR)	создавать приложения VR и AR для промышленности	практическими навыками применения VR и AR в промышленном производстве
Эталонный	Основной и дополнительный материал, предусмотренный компетенциями ПК-8, ПК-13, ПК-16 и ПК-19 без ошибок и погрешностей	**Умеет в полном объеме** … создавать приложения VR и AR для промышленности	всеми навыками применения VR и AR в промышленном производстве, демонстрируя их не только в стандартных ситуациях, но и при решении нестандартных задач
Продвинутый	основной материал, предусмотренный компетенциями ПК-8, ПК-13, ПК-16 и ПК-19, без ошибок и погрешностей	**Умеет с незначительными погрешностями** … создавать приложения VR и AR для промышленности	основными навыками применения VR и AR в промышленном производстве, демонстрируя их в стандартных ситуациях, в том числе при решении дополнительных задач
Пороговый	большинство основных понятий, изучаемых в	**Умеет с погрешностями** …	некоторыми основными навыками

	рамках дисциплины	создавать приложения VR и AR для промышленности	применения VR и AR в промышленном производстве, демонстрируя их в стандартных ситуациях

Критерии выставления оценки студенту на зачете

Порядок начисления рейтинговых баллов по предмету:
выполнение всех лабораторных работ – 100 баллов.

Баллы (рейтинговой оценки)	Оценка экзамена	Требования к сформированным компетенциям
85-100	«отлично»	Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно связывать теорию с практикой, свободно справляется с задачами, вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, использует в ответе материал монографической литературы, владеет разносторонними навыками и приемами выполнения лабораторных работ.
70-84	«хорошо»	Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении лабораторных работ вопросов и задач, владеет необходимыми навыками и приемами их выполнения.
50-69	«удовлетворительно»	Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала, испытывает затруднения при выполнении лабораторных работ.
0-49	«неудовлетворительно»	Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала по, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы. Как правило, оценка «неудовлетворительно» ставится студентам, которые не могут продолжить обучение без дополнительных занятий по соответствующей дисциплине.

Правила аттестации для студентов, не набравших необходимый минимум баллов по дисциплине

Если студент, в ходе изучения дисциплины набрал 50 и более баллов, то он имеет право на выставление зачета с соответствующей оценкой без его сдачи.

Если студент набрал менее 50 баллов, то он должен сдавать зачет (выполнение дополнительной лабораторной работы). Данный тест оценивается в диапазоне от 0 до 30 баллов. Полученные баллы суммируются к уже набранным и студенту выставляется итоговая оценка.

Текущий контроль
Устный опрос

Тема 1

1. Предпосылки, история, области применения систем виртуальной реальности.
2. Базовые понятия и определения технологий виртуальной и расширенной реальности.
3. Этапы и технологии создания систем VR, структура и компоненты.
4. Функциональные возможности современных приложений и сред с иммерсивным контентом.
5. Сферы применения и использования технологий виртуальной и расширенной реальности.
6. Составляющие иммерсивного контента.
7. Идея и сценарий для приложений разного уровня погружения в виртуальное пространство.
8. Обзор современных 3D-движков. Сравнительный анализ.
9. Разница между AR, Virtual Reality (VR) и Mixed Reality.
10. Оборудование для разработки XR приложений.

Тема 2

1. Классификация устройств визуализации и взаимодействия для иммерсивных сред.
2. Устройства визуализации виртуальных объектов: VR шлемы, очки дополненной реальности, панели и мониторы для отображения виртуальных объектов.
3. Устройства визуализации виртуальных объектов: очки дополненной реальности.
4. Устройства визуализации виртуальных объектов: панели и мониторы для отображения виртуальных объектов.
3. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: платформы.
4. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: системы трекинга головы.
5. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: системы трекинга глаз.
6. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: системы трекинга движений тела.

7. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: перчатки.
8. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: 3D контроллеры.
9. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: устройства с обратной связью.
10. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: датчики.

Темы 3, 4, 5

1. Маркерные технологии дополненной реальности.
2. Сферы применения дополненной реальности. Ограничения технологии дополненной реальности.
3. Технологии дополненной реальности. Архитектура приложений дополненной реальности.
4. Распознавание образов. Методы распознавания образов.
5. Типы задач распознавания образов.
6. Сенсоры, манипуляторы, устройства распознавания жестов.
7. Основы работы с SDK Unity 3D.
8. Создание VR-приложения с использованием SDK Unity.
9. Использование библиотеки OpenCV для разработки приложений расширенной реальности.
10. Использование платформы Vuforia для создания приложений расширенной реальности с полисенсорным управлением.

Зачет

Вопросы к зачету:

1. Базовые понятия и определения технологий виртуальной и расширенной реальности.
2. Этапы и технологии создания систем VR, структура и компоненты.
3. Сферы применения и использования технологий виртуальной и расширенной реальности.
4. Составляющие иммерсивного контента.
5. Основы работы с SDK Unity 3D.
6. Классификация устройств визуализации и взаимодействия для иммерсивных сред.
7. Распознавание образов. Методы распознавания образов.
8. Устройства визуализации виртуальных объектов: VR шлемы, очки дополненной реальности, панели и мониторы для отображения виртуальных объектов.
9. Устройства взаимодействия с виртуальными объектами в иммерсивных средах: системы трекинга головы, глаз, движений тела; перчатки, 3D контроллеры, устройства с обратной связью, платформы, датчики.
10. Использование платформы Vuforia для создания приложений расширенной реальности с полисенсорным управлением