Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Создание 2D-игры в жанре «roguelike» с использованием Unity Engine

Урбанский Денис Андреевич

Отчёт по учебной практике в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к. т. н., Ю. В. Литвинов

Санкт-Петербург
2023

Оглавление

Введение

В 2011 году вышла компьютерная игра Magicka, и всего за год её продажи достигли отметки в 1 300 000 копий[1]. Её отличительной особенностью была механика ведения боя: вместо обычного меню выбора заклинаний в распоряжении игрока было восемь стихий, до пяти из которых он мог использовать в колдовстве и, как следствие, нанесении вреда противникам или своему персонажу.

Также в последнее время все большей популярностью пользуется жанр игр “roguelike”. Он предполагает случайную генерацию уровней и не предусматривает продолжение игры с последнего сохранения в случае смерти персонажа. Иногда к этому добавляют мета-прогрессию – возможность разблокировки нового контента и/или улучшения персонажа между попытками прохождения.

Учитывая успех Magicka и рост популярности этого жанра, было бы интересно совместить их ключевые особенности, что привлекло бы умеренно казуальных игроков, открытых к экспериментам.

На основании вышесказанного создание продукта, включающего в себя аналог боевой системы Magicka и свойства жанра «roguelike», как минимум обернулось бы любопытным опытом, а как максимум — коммерческим успехом.

В результате учебной практики планируется реализовать прототип видеоигры, подходящей этим критериям, при помощи игрового движка Unity Engine, поскольку его основным языком является C#, с которым автор уже знаком, и на нём опубликовано больше всего продуктов на площадке Steam[2].

[1] К сожалению, ссылка на официальную страницу Стокгольмской Конвенции, упоминаемой по этому адресу, недоступна. URL: https://www.gamewatcher.com/news/2012-19-01-magicka-sells-1-3-million-copies-since-launch-new-expansion-revealed (дата доступа: 14 апреля 2023 г.).

[2] https://steamdb.info/tech/ (дата доступа: 13 апреля 2023 г.).

Дата сборки: 14 апреля 2023 г.

1. Постановка задачи

1.1. Цель

Целью данной учебной практики является создание прототипа двумерной игры в жанре «roguelike» с использованием Unity Engine. Для ее выполнения были поставлены следующие задачи:

1.2. Задачи на осенний семестр:

1. Обзор аналогов.
2. Проектирование базовой логики игры.
3. Реализация основных механик: простой ИИ противников, создание заклинаний из имеющихся у игрока сфер.
4. Разработка плана апробации.

1.3. Задачи на весенний семестр:

1. Реализация пользовательского интерфейса.
2. Добавление новых видов противников, сфер и заклинаний.
3. Реализация статус-эффектов: получение периодического урона при горении, замедление при воздействии холодом и т. п.
4. Добавление звуков, анимаций и частиц.
5. Проведение апробации.

2. Обзор

Для сравнения были выбраны игры, либо по своему игровому процессу напоминающие Magicka, либо относящиеся к жанру “roguelike”. Целью обзора является выделение особенностей таких продуктов и их подхода к механике ведения боя.

2.1. Noita

Noita[3] — 2D-боевик с пиксельной графикой и видом сбоку, в котором каждый пиксель моделируется физически. В начале каждой партии в распоряжение игрока даются два посоха: стандартный с двумя простыми заклинаниями-снарядами, который использует ману, и второй с одним случайным заклинанием с ограниченным количеством применений. Принцип работы каждого посоха зависит в основном от вложенных в него заклинаний, причём что посохи, что заклинания можно либо найти в процедурно генерируемом мире, поделённом на уровни, либо купить в магазине между ними. Однако “наполнять” свои орудия смерти получится только в этих же магазинах, что двояко: с одной стороны, отсутствует возможность быстро подстраиваться под ситуацию, с другой — не нужно каждый раз выбирать состав заклинания, и игрок ограничен только своей маной и временем перезарядки посоха. Концов у Noita несколько, но самый очевидный из них достигается спуском на самый нижний уровень пещер и победой над обитающим там монстром.

Игра отличается непредсказуемостью, которая вкупе с постоянной смертью делает её сложной.

2.2. Nine Parchments

Nine Parchments[4] — твин-стик шутер (как и Magicka, что примечательно) с элементами RPG, ориентированный на многопользовательский режим до четырёх человек, хотя и с возможностью поиграть од-

[3] https://store.steampowered.com/app/881100/Noita/ (дата доступа: 13 апреля 2023 г.).

[4] https://store.steampowered.com/app/471550/Nine_Parchments/ (дата доступа: 13 апреля 2023 г.).

ному. Перед каждым новым прохождением игрок выбирает одного из девяти персонажей (на момент первого запуска доступно только два из них), каждому из которых доступно по три заклинания, которые имеют одну из пяти стихий и различное поведение: от простых снарядов и пульверизаторов до проклятий, наносящих периодический урон и передающихся ближайшим врагам при смерти носителя. Арсенал пополняется по мере игры вплоть до девяти заговоров, а новых персонажей и посохи, дающие бонус к некоторым характеристикам, можно получить, выполняя задания с определёнными условиями.

2.3. Использованные технологии

В качестве движка был выбран Unity Engine, поскольку в Сети доступно множество обучающих материалов по нему; и, как было сказано ранее, он использует C#, известный автору, и имеет большую популярность у авторов, опубликовавших свои игры в Steam. Проект в Unity состоит из сцен, включающих в себя игровые объекты (GameObjects), к которым привязаны компоненты (Components). Компоненты определяют множество аспектов игровых объектов: от положения и размера до испускаемых ими частиц и скриптов, задающих их поведение. Все скрипты представляют собой код, написанный на C#, и те из них, которые требуется привязать к игровым объектам, должны наследоваться от базового класса Unity — MonoBehaviour. Для этого класса определены методы жизненного цикла объекта, которые исполняются каждый отрисованный кадр, пока объект активен, или при его уничтожении, или при активации объекта, или в других случаях[5].

Некоторые скрипты могут наследоваться от ScriptableObject — другого класса, предоставляемого Unity для хранения информации. Такие скрипты могут использоваться для хранения информации, которая должна переходить между сценами, например, для экземпляров оружия с уроном, легко задаваемым через окно редактора, или для сохранения прогресса игрока.

[5] https://docs.unity3d.com/Manual/ExecutionOrder.html (дата доступа: 13 апреля 2023 г.).

3. Реализация

В первом семестре работа была ориентирована на реализацию механики создания заклинаний и базовую игровую логику.

3.1. Что было сделано

- введена возможность двигаться (с помощью клавиш WASD) и здоровье игрока;
- спроектирована базовая логика противников: здоровье, движение в сторону игрока;
- создана система создания и произношения заклинаний, зависящих от сфер, входящих в их состав: так, например, заклинания, состоящие только из сфер-снарядов или только из сфер-пульверизаторов будут выпускать снаряд или распылять частицы соответственно, но если их объединить, то получится снаряд, при столкновении взрывающийся и наносящий урон по площади;
- добавлены две сферы: огненная — пульверизатор и каменная — снаряд.

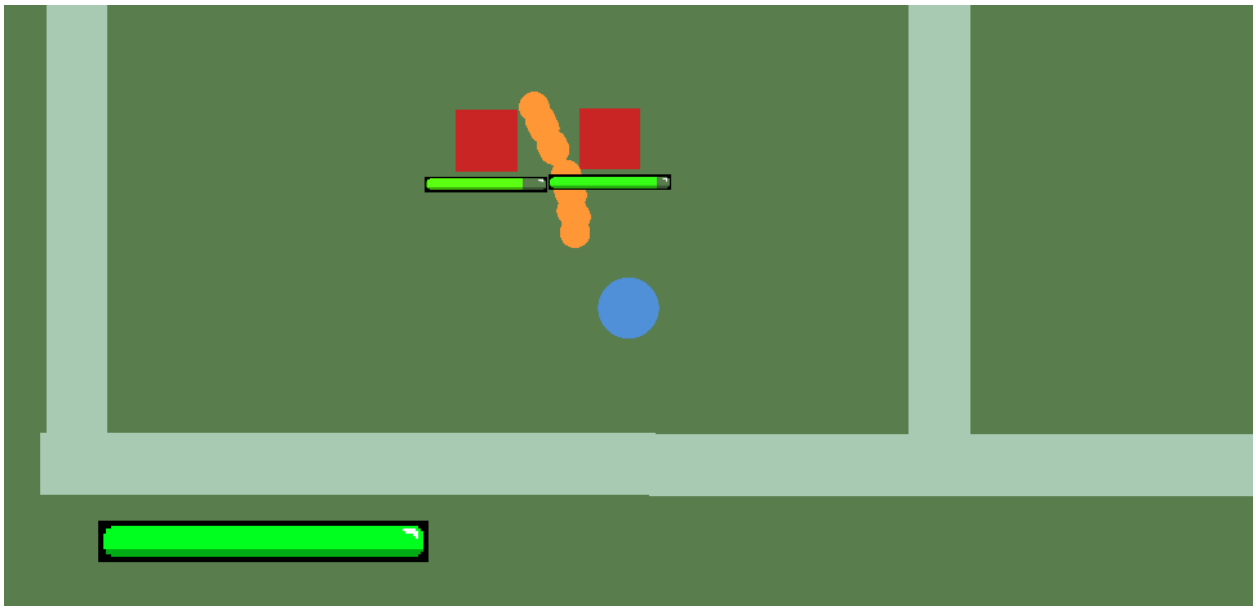

Рис. 1: Скриншот из игры

3.2. Портрет типичного игрока

Владимир, 16 лет. Учится в профильном техническом классе лицея. Чтобы отвлечься от учебы, ищет игру, в которой было бы важно разумно использовать предоставленный арсенал в зависимости от окружения, но которая не была бы слишком требовательна к игроку. Из-за учёбы у него нет времени на затяжные партии с возможностью переигрывать некоторые моменты несколько раз, и по той же причине он не хотел бы разбираться в новой видеоигре и приветствовал реиграбельность.

3.3. Архитектура

В архитектуре проекта был реализован компонентный подход, то есть для каждого игрового объекта поведение задаётся большим количеством компонентов, выполняющих отдельные функции. Это облегчает рефакторинг и позволяет использовать одни и те же фрагменты кода в разных объектах.

3.3.1. Перемещение и здоровье игрока

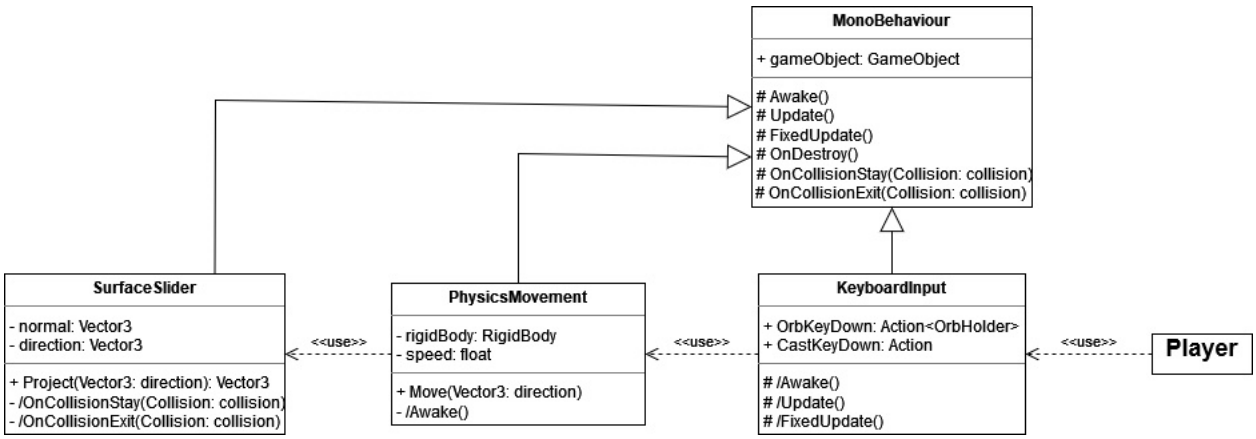

Рис. 2: KeyboardInput и скрипты, задающие перемещение игрока (MonoBehaviour взят из библиотеки Unity)

KeyboardInput передает направление, по которому хочет переместиться игрок в PhysicsMovement, отвечающий за само движение.

PhysicsMovement, в свою очередь, использует SurfaceSlider в случае столкновения игрового объекта с другими. SurfaceSlider проецирует желаемое направление движения на плоскость соприкосновения двух коллайдеров, не давая тем самым игроку входить, например, в стену.

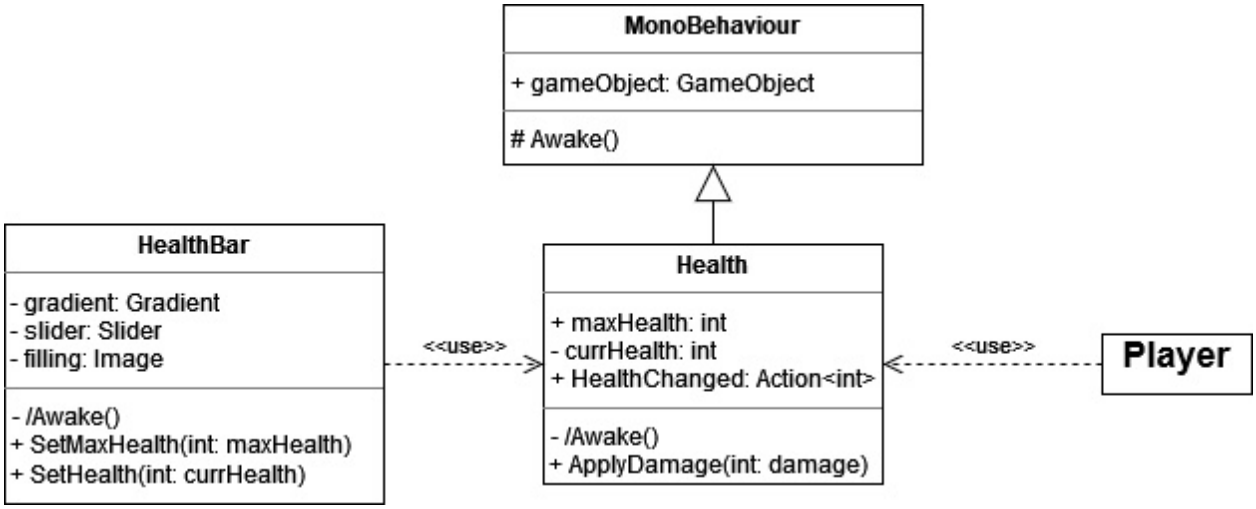

Рис. 3: Скрипты, задающие здоровье игрока и его отображение в интерфейсе

Скрипт Health отвечает за текущее и максимальное здоровье, а также в нем определен метод получения урона. HealthBar уведомляется о каждом изменении здоровья и изменяет показания — наполненность и цвет от зеленого до красного — полосы здоровья в соответствии с текущими данными.

3.3.2. Сферы и создание заклинаний

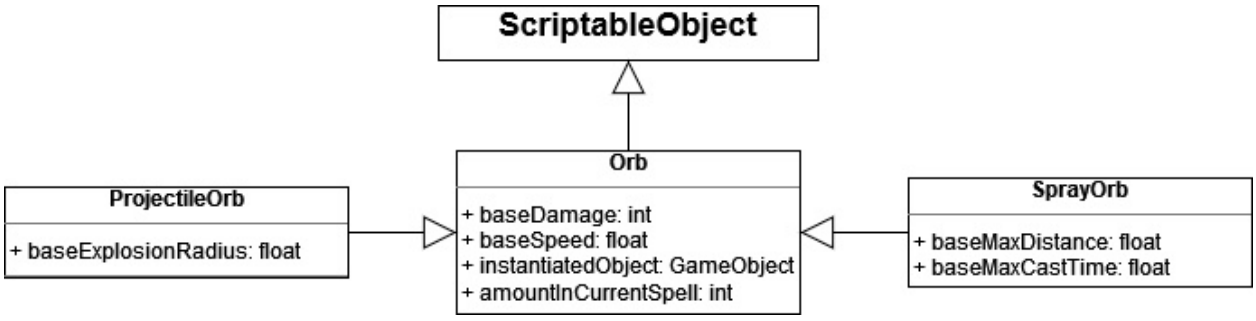

Рис. 4: Классы, отвечающие за сферы и их характеристики (ScriptableObject взят из библиотеки Unity)

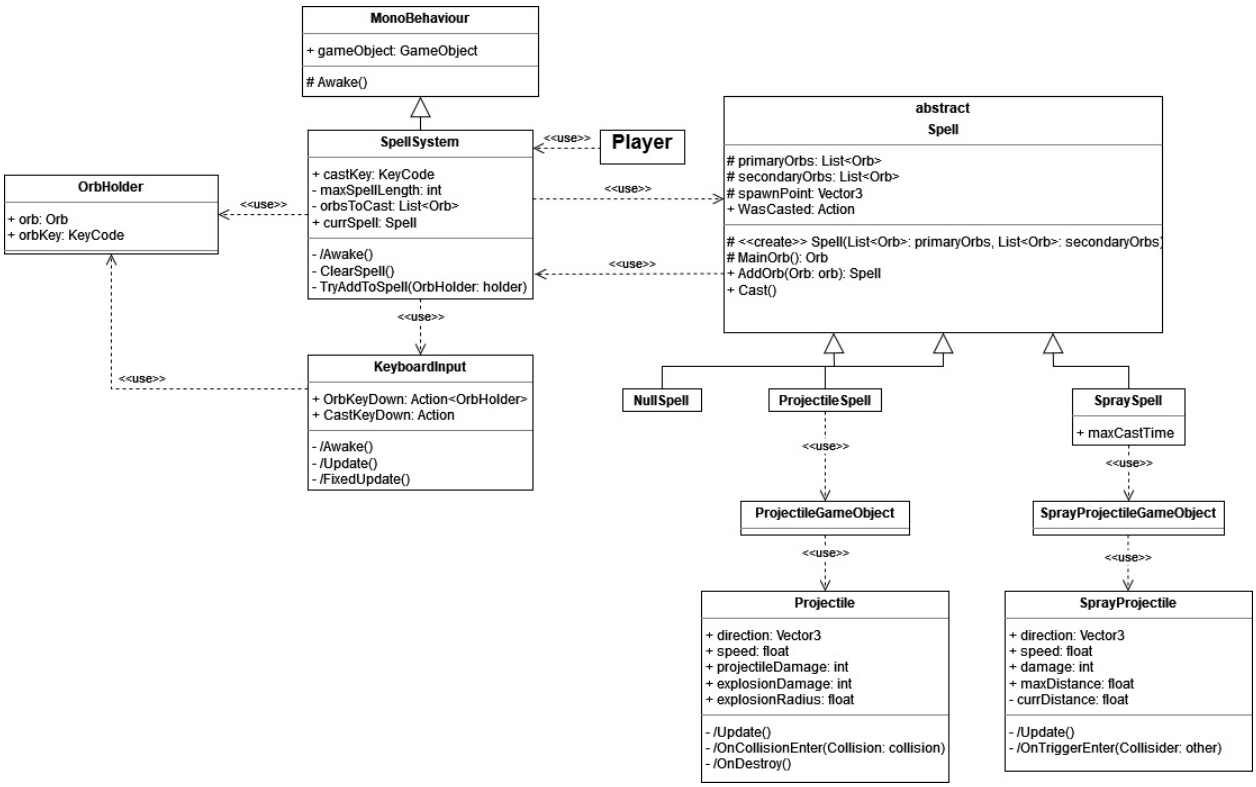

Рис. 5: Классы и скрипты, связанные с созданием заклинаний

Существует два типа сфер: создающие снаряд и создающие пульверизатор; общие для всех сфер характеристики задаются родительским классом Orb, наследуемым от ScriptableObject, в то время как уникальные данные для каждого типа сферы хранятся в дочерних классах ProjectileOrb (для сфер-снарядов) и SprayOrb (для сфер-распылителей)

У игрового объекта игрока есть несколько компонентов класса OrbHolder — это хранилище для сфер персонажа. Для каждого из них задана клавиша добавления своей сферы в заклинание. А созданием новых чар занимается SpellSystem: при попытке добавления сферы он проверяет, не превышено ли максимальное количество, и после этого в зависимости от типа текущего заклинания и типа добавляемой сферы возвращает новое заклинание одного из этих двух типов.

Стоит отметить, что абстрактный класс Spell не наследуется от MonoBehaviour.

Точка появления получившегося заклинания и его направление движения рассчитываются, исходя из положения в игровом мире курсора мыши и игрока.

В зависимости от типа заклинания логика создаваемого им объекта меняется: для снарядов создан компонент Projectile, который хранит направление движения, скорость, урон при столкновении, урон при взрыве и радиус взрыва, а также обеспечивает перемещение и взрыв при столкновении с чем-либо; для распылителей реализован компонент SprayPtojectile, уничтожающий снаряды при прохождении ими максимального расстояния или столкновении с противниками.

3.3.3. Поведение противников

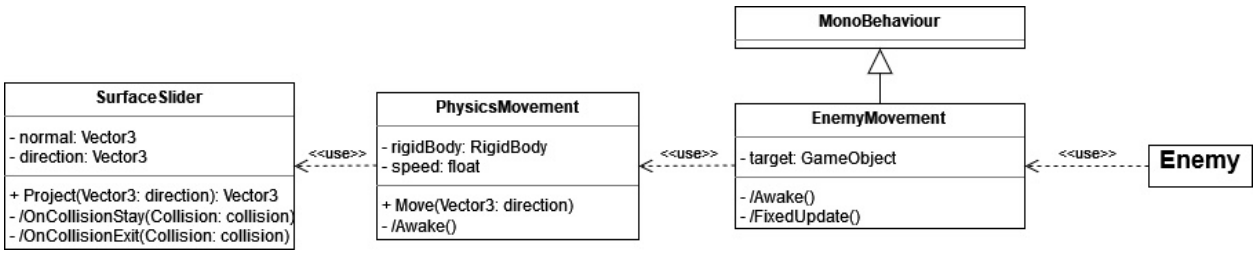

Рис. 6: Скрипты, отвечающие за перемещение противников

Для реализации движения противников были использованы те же PhysicsMovement и SurfaceSlider, что и для перемещения игрока, однако направление берется из EnemyMovement, который нацеливает врагов на игрового персонажа. Для реализации и отображения здоровья противников также были повторно использованы скрипты, работающие со здоровьем игрока.

4. Апробация

Апробация запланирована на весенний семестр и состоит в том, чтобы предложить друзьям, знакомым или другим потенциально заинтересованным лицам поиграть в игру и оценить её при помощи System Usability Scale [1].

Заключение

Итогом данной учебной практики послужили следующие результаты:

- проведен обзор аналогов;
- спроектирована базовая логика игры;
- реализованы основные механики;
- разработан план апробации.

Исходный код доступен по ссылке: https://github.com/EnderDodo/Orbitrator

Список литературы

[1] Wikipedia contributors. System usability scale — Wikipedia, The Free Encyclopedia. — 2023. — [Online; accessed 14-April-2023]. URL: https://en.wikipedia.org/w/index.php?title=System_usability_scale&oldid=1147509546.