отрасли настолько огромны, что ее можно сравнить с кинопроизводством. Учитывая скорость развития игровой индустрии велика вероятность того, что в скором времени она будет существенно опережать любые другие виды развлечений. Видеоигры оказывают большое влияние на потребителей и вовлекают в интерактивное окружение.

Разработанное игровое приложение «Ocean story» рассказывает историю героини Анны, которая неожиданно узнала о возвращении некогда пропавшей без вести сестры. Однако, сестра уже совсем не тот человек, которого помнит главная героиня. Игрока ждет увлекательное приключение с целью раскрыть тайны пропавшей сестры и разобраться со странными вещами, происходящими после её появления. Игровой процесс сопровождается отличным саундтреком, а концовка игры не оставит равнодушным даже самого требовательного игрока.

Игровое приложение было разработано при помощи одного из самых популярных и прогрессивных на данный момент игровых движков – Unreal Engine 4. Это позволило добиться баланса в графике и производительности. Игровая логика была написана на встроенном в движок языке программирования Blueprints, а также некоторые скрипты используют язык C++.

Благодаря современным играм в жанре adventure игрок может отвлечься от насущных проблем и с головой погрузиться в захватывающий мир приключений.

В. В. Суомалайнен, Е. В. Комракова
(*ГГТУ им. П. О. Сухого, Гомель*)

РЕШЕНИЕ ЗАДАЧИ «ПРОХОЖДЕНИЕ ЛАБИРИНТА» ПРИ ПОМОЩИ PYTHON

Задача нахождение пути в лабиринте напоминает задачи поиска в информатике. При реализации данного решения, можно применить различные алгоритмы поиска, но при этом, программная архитектура останется неизменной. За создание лабиринта отвечает класс *Maze,* который генерирует случайный лабиринт, имеющий выход. За вывод лабиринта отвечает метод *__str__*(). Проверка выхода из лабиринта проходит в методе *goal_test*(). Метод нахождения возможных направлений движения с выбранной точки – *successors*().

Поиск пути в лабиринте базируется на алгоритме поиска в глубину, который заходит настолько глубоко, пока не будет найдено новых точек для движения или выход из лабиринта.

Для алгоритма поиска необходимо реализовать стек, который будет содержать элементы лабиринта. Стек содержит методы: *push*() – помещает элемент на вершину стека; *pop*() – удаляет элемент с вершины данного стека и возвращает его; *empty*() – проверят стек на наличие элементов. Также необходимо определить класс *Node*, для отслеживания перехода от места к месту.

Во время работы алгоритм поиска в глубину (*DFS*) отслеживает две структуры данных: стек рассматриваемых мест, именуемый как *frontier* и набор просмотренных состояний – *explored.*

Если *DFS* завершается успешно, то вернётся объект типа *Node* в котором находится исходное состояние всего маршрута от начальной до конечной точки. И чтобы получить данный маршрут, необходимо двигаться от обратного, обращаясь к свойству *parent* у текущего *Node.* За это отвечает метод *node_to_path*(). Построение лабиринта с учётом успешного найденного пути реализовано в методе *mark*(). Для очистки лабиринта реализован метод *clear*().

В результате было получено приложение, способное генерировать лабиринты различной сложности, а также находить выход из различных лабиринтов посредством реализованного алгоритма. Для поиска решений возможна реализация любого алгоритма, главное, чтобы алгоритм возвращал соответствующий набор данных, необходимый программному решению.

Д. В. Тарасенко, Е. В. Комракова
(*ГГТУ им. П. О. Сухого, Гомель*)

СОЗДАНИЕ ЭКРАННЫХ ЭФФЕКТОВ В UNITY С ПОМОЩЬЮ РЕНДЕР-ТЕКСТУР

Один из наиболее впечатляющих аспектов изучения шейдеров является написание своих собственных полноэкранных эффектов. Их так же называют постэффектами или фулскрин-эффектами. Данные эффекты функционируют следующим образом: *Unity* рендерит изображение приходящее с камеры, отдаёт текстуру в эффект, который, используя шейдер на *GPU*, возвращает модифицированное изображение.