Санкт–Петербургский государственный университет

Математическое обеспечение и адмиистрирование информационных систем

Егорова Лада Владимировна

Реализация игры «Elevator»

Отчет по учебной практике

Научный руководитель:
к.т.н., доц. Ю.В. Литвинов

Санкт-Петербург
2020 г.

Содержание

1. Введение 3

2. Постановка задачи 4

3. Обзор аналогов 5

3.1. Существующие решения 5

4. Реализация 7

4.1. Архитектура 7

4.2. Персонажи на уровне 7

4.3. Движение лифта 9

4.4. Статистика на уровне и интерфейс. 9

4.5. Создание дома 10

5. Апробация среди пользователей 12

6. Заключение 13

Список литературы 14

Приложение 1 15

1. Введение

Игровая индустрия сейчас предлагает множество жанров игр для всех возрастов. Одним из самых популярных видов игр остаются гипер-казуальные игры — игры с очень простой механикой, в которые играют как дети, так и взрослые, чтобы скоротать время.

На летней школе, которая проходила в июле 2019 года, была создана студия под названием «NoPayStudio» для продолжения разработки игр после летнего периода. В марте этого года наша команда решила реализовать проект, предложенный продюсером, а именно игру про лифт, с задачей развозить людей по этажам. Этот проект был назван «Elevator».

Чтобы грамотно и быстро создать приложение, было необходимо обозреть существующие решения, а также разработать его архитектуру.

2. Постановка задачи

Целью работы является создание приложения на Android — игры про лифт, развозящий людей по этажам.

Проект выполнялся в команде, и задачами автора были:

1. Реализовать механику движения лифта;
2. Описать систему создания различных персонажей на уровне;
3. Добавить механизм развоза персонажей по этажам;
4. Реализовать приложение;

Задачи, общие для всей команды:

1. Выполнить обзор существующих решений и выбрать инструмент реализации;
2. Получить рабочую версию приложения;
3. Провести апробацию среди пользователей.

3. Обзор аналогов

3.1 Существующие решения

Одними из существующих симуляторов лифтера на Android являются Act On Elevator, Elevator Rescue и Lift EM All.

Act On Elevator. Игра-шутер для Nintendo, локация — дом. Лифт используется для перемещения как главного героя, так и его противников. Механика отличается от задуманной командой, но расположения лифта и дома во многом похожи.

Elevator Rescue. Цель игры — спасти людей от пожара в здании, посадив их в падающий лифт. Механика — нажатия на экран, когда один из людей находится в зоне досягаемости. На некоторых этажах находится динамит, который может взорваться внутри лифта, и все пассажиры в лифте будут потеряны.

Lift EM All. Игра с пиксельной графикой про лифт, который развозит пассажиров по этажам. По механике похожа на то, что было задумано студией, однако команда выбрала трёхмерный стиль реализации проекта.

Также был найден ряд игр-головоломок на побег из лифта, но их механика очень сильно отличается от идеи команды.

Таким образом, команда сделала вывод, что игра с 3D-графикой может быть интересна широкой аудитории, и начала разработку. Инструментом реализации был выбран движок Unity, а языком — C#.

Работа с Unity. Unity — движок, обладающий собственным редактором с набором инструментов. Сцена — основное окно редактора, на котором размещаются объекты. Объекты могут обладать разными свойствами, физическими и не только и взаимодействовать между собой.

Чтобы добавить в игру 3D объект, можно воспользоваться встроенными в Unity моделями, а можно использовать свои. Общий алгоритм не

поменяется — 3D модель, перенесенная на сцену, станет объектом, свойства которого разработчик вправе изменять.

Поведение объектов регулируется скриптами, то есть файлами с кодом, написанном в нашем случае на C#. Как правило, в одном скрипте описан только один класс. Для того, чтобы код работал с объектами, используется пространство имён UnityEngine, а класс в нём наследуется от класса MonoBehaviour. Написанный скрипт переносится на объект как свойство и объект начинает его исполнять при запуске игры.

Метод Update() вызывается один раз в квант времени. С его помощью можно обрабатывать пользовательский ввод, отслеживая, какая клавиша нажата, и выполнять инструкции, написанные разработчиком.

4. Реализация

4.1 Архитектура

Архитектура приложения проиллюстрирована диаграммой (см. Приложение 1. 1) В Unity игра состоит из сцен, между которыми осуществляются переходы. В «Elevator» стартовая и основная сцены соединены в одну для визуализации игры до нажатия кнопки «Start». Так как действия по запуску и перезапуску игры похожи, был создан один класс Restart. Он отвечает за очистку прогресса на уровне (прогресс очищается в трёх классах — Elevator (очищение человечков в лифте), House (на этажах) и Counter (обнуление счётчиков уже созданных и потерянных персонажей)) и активацию и деактивацию стартовой кнопки (если нажали на неё, она исчезает и появляется, когда уровень будет пройден).

4.2 Персонажи на уровне

Модели человечков были взяты из ассета для Unity, разработчику нужно было их добавить. Персонажи перевозятся с помощью лифта, который содержит определённое фиксированное количество слотов (8). В игре есть три класса пассажиров, отличающиеся занимаемым пространством в лифте, — маленькие, средние и большие (1, 2 и 3 слота соответственно). Персонажи создаются на уровне не одновременно, а с заданной периодичностью (по умолчанию каждые три секунды). Для этого используются так называемые корутины (Coroutines) — C#-итераторы, которые могут выполяться в цикле через определённые интервалы времени (с помощью выражения yield return WaitForSeconds(0.1f) — здесь время ожидания будет составлять примерно 100 миллисекунд)[2]. Для создания персонажей используется метод Passengers() класса PassengerController. Зная их общее число на уровне, он создаёт список из объектов, каждый раз с новым классом Person, Child или FatGuy (в зависимости от типа). Далее вызывается статический метод StartCoroutine, начинающий работу указанной корутины, в этом случае Creator: StartCoroutine(Creator(interval, list)). Для всех позиций из списка производятся следующие действия внутри корутины:

1. Ожидание в течение числа секунд, указанного в переменной interval;

2. Создание персонажа на сцене на нужном «этаже», то есть дочернем объекте объекта House (считывается свойство StartFloor из класса, унаследованного от Person (который был инициализирован в списке)).

3. Добавление таймера над персонажем (его координаты + 1.5f * Vector3.up). Vector3.up — вектор с координатами (0, 1, 0). Таким образом удобно задаётся позиция объекта таймера.

Каждый человечек — отдельный объект, которому нужно задавать поведение. Но так как типы персонажей отличаются, им задаются соответствующие скрипты. Все они унаследованы от одного скрипта «Passenger», определяющего основную логику поведения: заходить в лифт, если он на этаже, выходить из него, когда он приезжает на нужный этаж, анимацию ходьбы и поведение при столкновении с прозрачным барьером. Каждый скрипт создаётся своим образующим объектом, что является реализацией паттерна «Фабричный метод»[3].

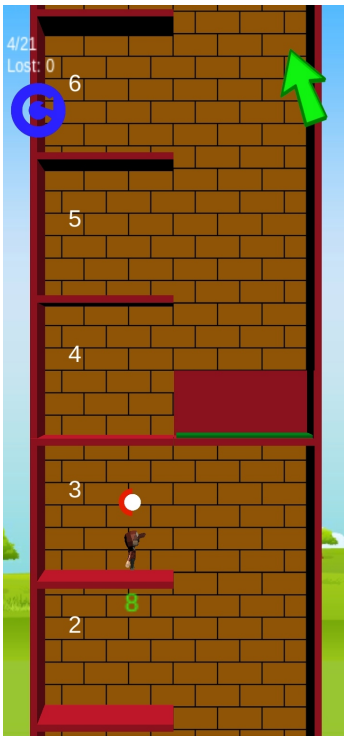

Рис. 1: Таймеры на персонажах.

Индикаторы на персонажах. У каждого пассажира есть номер этажа, на который ему нужно, отображающийся под ним, и таймер (см. рис. 1), находящийся над ним. Таймер иллюстрирует терпение пассажира, по умолчанию каждый персонаж ждёт 15 секунд. Он имеет формулу круглого прогрессбара для наглядности и стиля. Когда он достигает нуля, человечек исчезает и счётчик потерянных пассажиров увеличивается на 1. Отсчёт ведётся в классе Passenger с помощью метода FixedUpdate(). Если был достигнут предел ожидания, то есть таймер обнулился или стал очень близок к нулю, вы-

зывается метод DestroyFromFloor(), удаляющий его из числа объектов на этаже, а затем метод Destroy(), удаляющий объект.

4.3 Движение лифта

Особое внимание было уделено механике движения лифта на уровне. Требовалось добиться плавного скроллинга, аналогичного прокрутке ленты новостей в социальных сетях на экранах мобильных устройств. Требовалось, чтобы лифт по возможности оставался в центре экрана, и иллюзия его движения достигалась за счёт движения дома. Таким образом, все перемещения лифта разделились на три вида:

1. Скроллинг для передвижения лифта игроком, осуществляется за счёт считывания позиций касаний, вектора их разности и перемещения дома на этот вектор;

2. Если игрок отпускает палец, а лифт находится между этажами, ему необходимо сначала подъехать к ближайшему, а затем высадить пассажиров и посадить новых. Поэтому был добавлен отдельный вид движения: сначала просчитываются координаты лифта до этажей и выбирается ближайший, вычисляется вектор, необходимый для того, чтобы выровнять лифт, а затем дом перемещается на этот вектор.

3. Третий вид описан с учётом того, что игрок может не догадаться отпустить палец, и в таком случае лифт никогда не сможет посадить пассажиров. Поэтому, если лифт долгое время находится рядом с каким-то этажом, тоже срабатывает второй вид движения, лифт подъежает к этажу и высаживает человечков.

4.4 Статистика на уровне и интерфейс.

Статистика на уровне. В верхнем левом углу экрана игры ведётся статистика — число уже созданных персонажей/общее число персонажей на уровне. Также отдельно подсчитываются пассажиры, которых игрок не успел подобрать. Может случиться так, что пассажир появится на этаже, который не видно в текущий момент. Для этого предусмотрены

стрелки, показывающие соответственно вверх и вниз, чтобы игрок лучше ориентировался в игре. Класс PassengerController хранит список персонажей на экране весь игровой процесс. Каждый раз при вызове метода Update() он подсчитывает, сколько персонажей из списка имеют свойство Upper, а сколько Lower. Эти свойства обновляются в классе Passenger у каждого из человечков. Мы получаем преобразование из позиции в мировом пространстве в позицию на экране с помощью статического метода класса Camera WorldToViewportPoint. Если координаты x и y по модулю меньше 1, то объект видно на экране (он попадает в видимый прямоугольник), если нет, то смотрим, у больше 0 (объект выше) или меньше (ниже). Если число персонажей со свойством Upper или Lower не равно 0, то активируем верхнюю или нижнюю стрелку соответственно (могут быть активны обе стрелки сразу).

Интерфейс. Кнопки старта (см. рис. 2). и рестарта, — часть пользовательского интерфейса, — являются изображениями в формате .png. Они добавляются в проект Unity, а потом разработчик с помощью редактора преобразует их в «спрайты» — особый вид объектов-изображениямй, которые могут взаимодействовать с другими объектами на сцене.

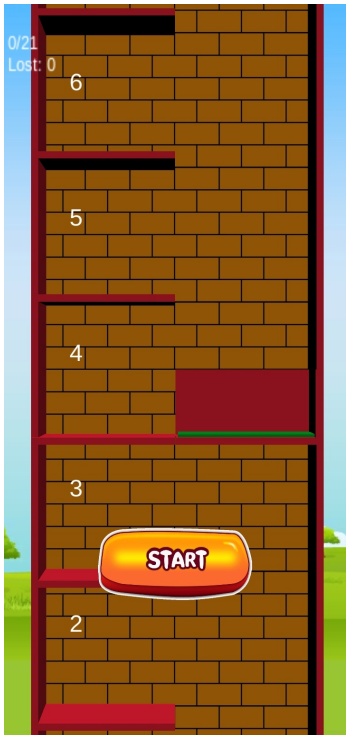

Рис. 2: Стартовый экран.

4.5 Создание дома

Все этажи в доме создаются из одного заранее заготовленного «префаба» — модели объекта, которая может быть перемещена на сцену, чтобы стать объектом. Можно указать необходимое число этажей в доме, и они будут созданы друг под другом без смещений и интервалов (это обеспечивается тем, что каждый этаж обладает свойством «Layout Element», то есть быть частью сетки с предусмотренными интервалами). На каждом этаже находится прозрачный «барьер» для избежа-

ния ситуации, когда пассажир идёт к приехавшему лифту, а он неожиданно уезжает на другой этаж. Барьер останавливает пассажира и заставляет вернуться обратно к месту ожидания, то есть позиция персонажа устанавливается так, как будто он создаётся заново (дочерней к этажу с учётом других пассажиров).

5. Апробация среди пользователей

К сожалению, из-за жёстких временных рамок и сложностей, связанных с публиацией игры в Play Market, команда была лишена возможности получить статистику о ней на основе отзывов. Поэтому было принято решение протестировать приложение с помощью друзей и знакомых, предложив им поиграть и оставить отзыв.

Некоторые из отзывов: «Интересная задумка, может затянуть, раньше такой не видела... Сначала не поняла, что вообще надо делать))(нет никакого вступления) + нужно какое-то постепенное усложнение, больше уровней сложности»

«Видно, что много работы проделано, продолжайте развиваться, держим кулачки!!... Хотелось бы в начале игры какое-то пояснение о том, что нужно делать, а то до нас дошло, что это реально elevator, только через секунд десять :D человечки какие-то странные! что забирай, что не забирай их, никакого прогресса или регресса в статистике это не добавляет??? это странно...»

Средняя оценка приложения — 4,0. Людям нравится идея игры, но многим хотелось бы получить увеличение уровня сложности и интуитивности интерфейса. Таким образом, был сделан вывод, что приложение может пользоваться большим спросом.

6. Заключение

Были достигнуты следующие результаты:

1. Реализована механика движения лифта;
2. Описана система создания различных персонажей на уровне;
3. Добавлен механизм развоза персонажей по этажам;
4. Реализовано приложение.

Командой были:

1. Обозрены существующие аналоги и выбран инструмент для реализации;
2. Получена рабочая версия приложения;
3. Проведена апробация среди игроков.

Список литературы

[1] Unity-Manual: Unity User Manual (2019.3) //— (дата обращения: 10.04.2020). https://docs.unity3d.com/Manual/index.html

[2] Coroutines — Unity Manual //— (дата обращения: 03.05.2020). https://docs.unity3d.com/ru/2018.4/Manual/Coroutines.html

[3] Фабричный метод (Factory Method) | Паттерны в C# и .NET //— (дата обращения: 20.04.2020). https://metanit.com/sharp/patterns/2.1.php

Приложение 1

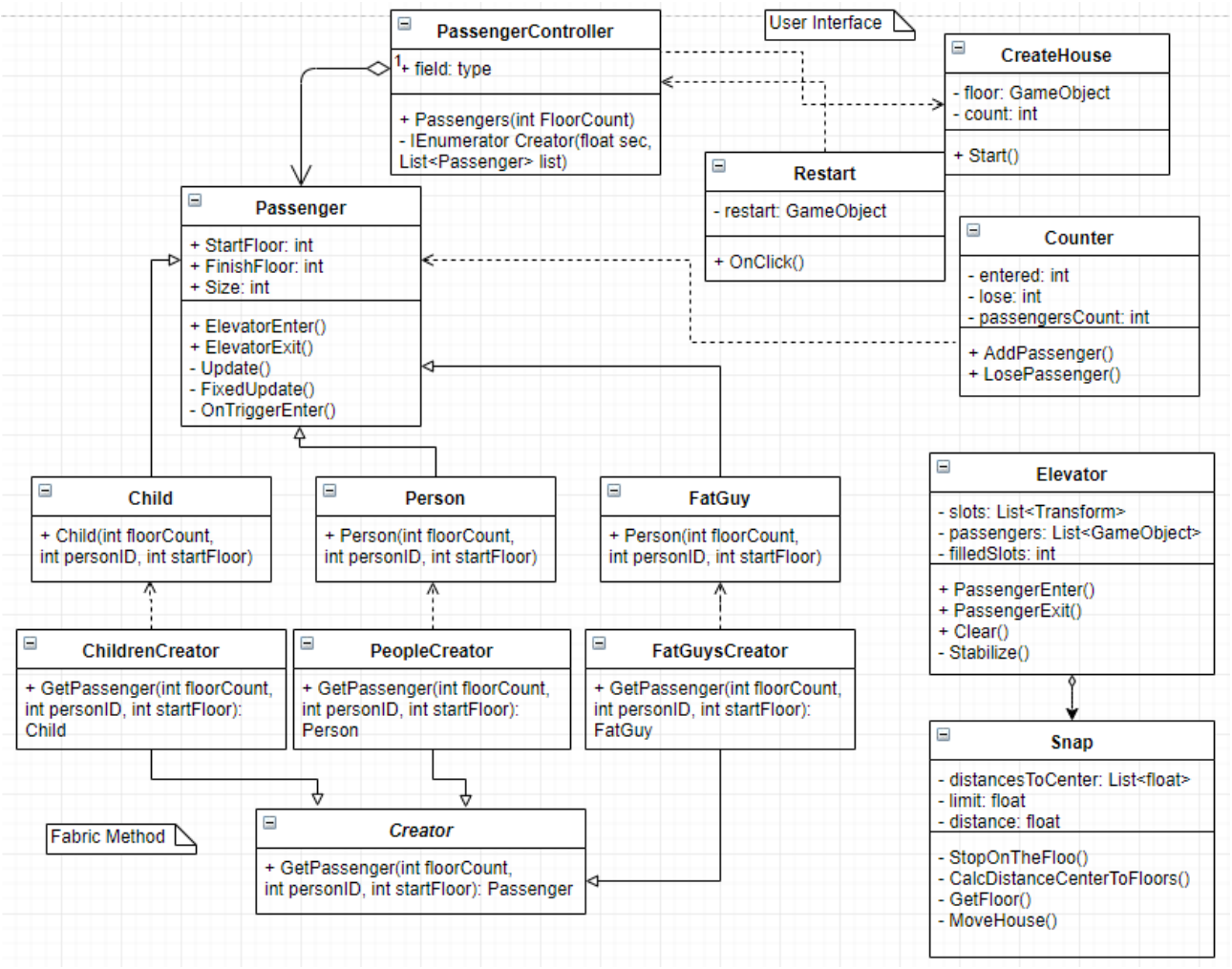

Рис. 3: Архитектура приложения