УДК 004.413.5

студ. А.А. Сосновский
Науч. рук. доц., к.ф.-м.н. Н.Н. Буснюк
(кафедра информационных систем и технологий, БГТУ)

ПРОГРАММНОЕ СРЕДСТВО LOGIC BLOCKS ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ANDROID

Скорость разработки и выпуска программных продуктов в современном мире играет значимую роль, так как конкуренция на рынке мобильных приложений (и не только мобильном рынке) высока, и то что не реализовано сейчас, может быть выпущено в ближайшее время. Кроме этого, необходимо минимизировать затраты производства. Поэтому, необходимо как можно быстрее внедрять новые функциональные возможности, получать от пользователей обратную связь и улучшать приложение. Этот круг, «разработка – тестирование – улучшение», разработчик проходит большое количество раз. Чем быстрее придет обратная связь от пользователей, тем большей статистикой будет обладать разработчик, и на основании этих данных сможет вести дальнейшую разработку программного продукта в ту или иную сторону.

Прототипирование является неотъемлемой частью разработки ПО, так как позволяет опробовать новые идеи (функциональные возможности, правки интерфейса и прочее), избегая больших временных и материальных затрат, сфокусировав внимание на главных деталях.

Прототип – это результат процесса прототипирования. Прототипы бывают разные: начиная от пользовательского интерфейса и заканчивая деревянной моделью кораблика. При разработке ПО обычно прототипируют ту часть ПО, которую необходимо “потрогать в живую”, то есть испытать на конечном пользователе. Это дает понятие разработчику о том, как пользователь воспринимает данную идею, как изменяется его поведение.[1]

В качестве примера прототипирования рассмотрим проект Logic Block.

Logic Blocks – это мобильная логическая игра, в которой пользователю предстоит проходить уровни, решая уровень из блоков, то есть перемещая блоки таким образом, чтобы игровое поле опустошалось. Игровое поле представляет собой прямоугольную матрицу размером M x N. Блоки представляют собой квадраты разных цветов. Менять местами можно только соседние блоки и строго по

вертикали или горизонтали. Если блоки одного цвета образуют специальную комбинацию, то эти блоки исчезают. Кроме этого, на игровое поле действует «гравитация», то есть блоки, под которыми нет других блоков, опускаются вниз, максимум до первой строчки в матрице. Если столбец матрицы становится пустым, то он убирается, а столбы по обе стороны от него соединяются. Пример игрового уровня представлен на рисунке 1.

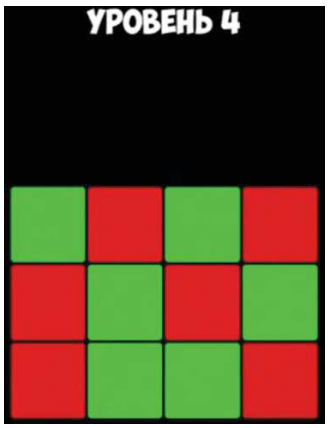

Рисунок 1 – Пример игрового уровня

Кроме основы игры, в современных играх реализовано огромное количество дополнительных функций, таких, например, как таблица рейтинга игроков (кто лучше, кто хуже), возможность поделиться результатом в социальных сетях, сохранение игрового прогресса и возможность продолжения игры с другого устройства, звуковое сопровождение, спецэффекты, достижения и масса других. Однако, не для всех игр это является обязательным, а точнее, коренным функционалом. Поэтому, в проекте Logic Blocks необходимо было реализовать только коренные функциональные возможности.

В Logic Blocks хотелось попробовать две идеи. Первая - игровые правила. Игровые правила – это сердце логических игр, так как от сложности и реиграбельности уровней (насколько интересно переигрывать один и тот же уровень), которые строятся на основе правил, зависит интерес к игре. Чтобы протестировать игровые правила, необходимо было разработать утилиту для генерации уровней (матриц), а также “решатель” уровней, чтобы убрать нерешаемые уровни из базы уровней.

Данный этап позволит создать скелет игрового процесса и проверить, насколько сложно или легко решать тот или иной уровень, а также нравится ли пользователю этот процесс или нет.

Вторая идея, которую хотелось попробовать, это добавить дополнительные параметры, которые могли бы мотивировать пользователя проходить последующие уровни. Например, это индикатор жизней, который уменьшается, если игрок не решил уровень.

При разработке идей необходимо пользоваться правилом «меньше делай, больше тестируй».

В ходе прототипирования иногда возникает необходимость улучшать некоторые параметры разрабатываемого функционала. В данном случае, необходимо было оптимизировать количество генерируемых уровней, так как при определенных параметрах их число было значительным (что требовало больших мощностных затрат и затрат на хранение), а количество необходимых из них уровней - малым. Однако не стоит доводить до идеала работу какой-либо функциональной возможности, так как это может потребовать огромное количество времени.

В примере выше с генерацией уровней изначально был применен алгоритм генерации последовательностей, который создавал цепочки определенной длины из комбинации нескольких цветов (размещения с повторениями). Данный алгоритм был оптимизирован следующими путями:

– если «шаблон» уровня повторялся, то его можно пропустить. К примеру, возьмем две строчки длинной 3 символа каждая, из двух символов - 1 и 0. Первая строка содержит 101, вторая 010. Эти строки принадлежат одному шаблону, так как если заменить символы второй строки на противоположные символы первой строки, то получится идентичная первой строка. Но если вторая строка содержит к примеру 100, то данная строка уже не подходит под шаблон первой строки. Таким образом, конечное число уровней можно уменьшить во столько раз, сколько цветов используется в уровне;

– если уровень содержит меньше символом одного типа, чем количество символов в минимальной комбинации, то данный уровень не решаемый и его следует пропустить;

– если уровень не решается «решателем» (программа, которая решает уровни автоматически), данный уровень пропускается.

На ряду с оптимизацией алгоритма генерации уровней был оптимизирован процесс записи конечных уровней в файл (к примеру, запись блока уровней в файл вместо записи каждого уровня отдельно), что дало прирост в производительности утилиты для генерации уровней.[2]

При разработке применялся принцип итерирования, который позволяет посмотреть на результат труда после выполнения небольшого объема задач. Это позволяет получать обратный отзыв от пользователей, что способствует улучшению качества приложения еще на этапе разработки, а не после выпуска.

Однако следует иметь ввиду, что доступ к прототипу необходимо предоставлять ограниченному кругу лиц, потому что исход тестирования может повлиять на пользователя как в хорошую сторону, так и в плохую. Например, если пользователю не понравится какая-либо новая функциональность, он может перестать пользоваться приложением. Поэтому стоит вначале протестировать функциональность на небольшом (но достаточном) количестве пользователей, а дальше, по результатам решить, стоит ли давать доступ всем остальным пользователям.

Чтобы минимизировать временные расходы на разработку прототипа, целесообразно выбирать те инструменты, с которыми уже приходилось работать. Это поможет избежать лишних проблем при решении какой-либо задачи. Проблемы могут быть из-за:

– небольшого количества документации и примеров для выбранного инструмента;

– наличия ошибок внутри инструмента;

– сложности выбранного инструмента.

В качестве игрового движка был выбран Unity3D, для написания программного кода – язык программирования C#. Для разработки утилиты, позволяющей создавать игровые уровни, был использован язык программирования C, так как он обладает большей скоростью выполнения и дает полный контроль над памятью, нежели C#.

В заключение стоит отметить, что процесс прототипирования можно применять на практике и в учебных целях, к примеру, при написании курсовых и дипломных работ, чтобы провести первичный анализ выбранной темы, а также получить реальные данные с производства при эксплуатации прототипа. Это поможет выбрать наиболее оптимальный путь развития задуманной идеи.

ЛИТЕРАТУРА

1 Прототипирование // ВИКИПЕДИЯ информационный портал. – [Электронный ресурс]. –2017. – Режим доступа: https://ru.wikipedia.org/wiki/Прототипирование. – Дата доступа: 03.05.2017.

2 Основные формулы комбинаторики // МатБюро информационный портал. – [Электронный ресурс]. –2017. – Режим доступа: http://www.matburo.ru/tv_komb.php. – Дата доступа: 03.05.2017.

УДК 004.85

Студ. В.К. Сенюк
Науч. рук.: ст. преп. Е.А. Блинова
(кафедра информационных систем и технологий, БГТУ)

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ РЕДАКТИРОВАНИЯ И ПОИСКА КРАТЧАЙШИХ ПУТЕЙ ОБХОДА ГРАФА

При решении многих задач старая привычка толкает нас рисовать на бумаге точки, изображающие людей, население пункты, процессы и т.д., и соединять эти точки линиями или стрелками, означающими некоторые отношения. Такие схемы встречаются всюду под разными названиями: в экономике - диаграммы организации, в физике - электрические цепи, сети коммуникаций и т.д. Однако все эти понятия сводятся к одному – граф. Граф – это совокупность непустого множества вершин и наборов пар вершин (ребер). Теория графов является предметом анализа всего, что содержит в себе большое количество сложных связей.

Существует достаточно большое количество видов графов: мультиграфы (с кратными ребрами), псевдографы (с петлями), ориентированные (связи – направленные дуги) и неориентированные графы (вершины связаны ребрами) и т.д.

Одними из основных операций над графом являются – поиск кратчайших путей из одной вершины в другую, а также поиск максимального потока. Для реализации этих операций, в зависимости от вида графа и его свойств используются различные методы обхода. В результате поиска в ширину находится путь кратчайшей длины в невзвешенном графе, т.е. путь, содержащий наименьшее число рёбер. В результате поиска в глубину находится лексикографически первый путь в графе. Алгоритм Дейкстры предназначен для нахождения кратчайшего расстояния от одной из вершин графа до всех остальных, работает только для графов без рёбер отрицательного веса. Алгоритм Левита также предназначен для поиска кратчайших путей в графе, но отличается от алгоритма Дейкстры тем, что работает и для графов, которые содержат в себе ребра и положительного и отрицательного веса.