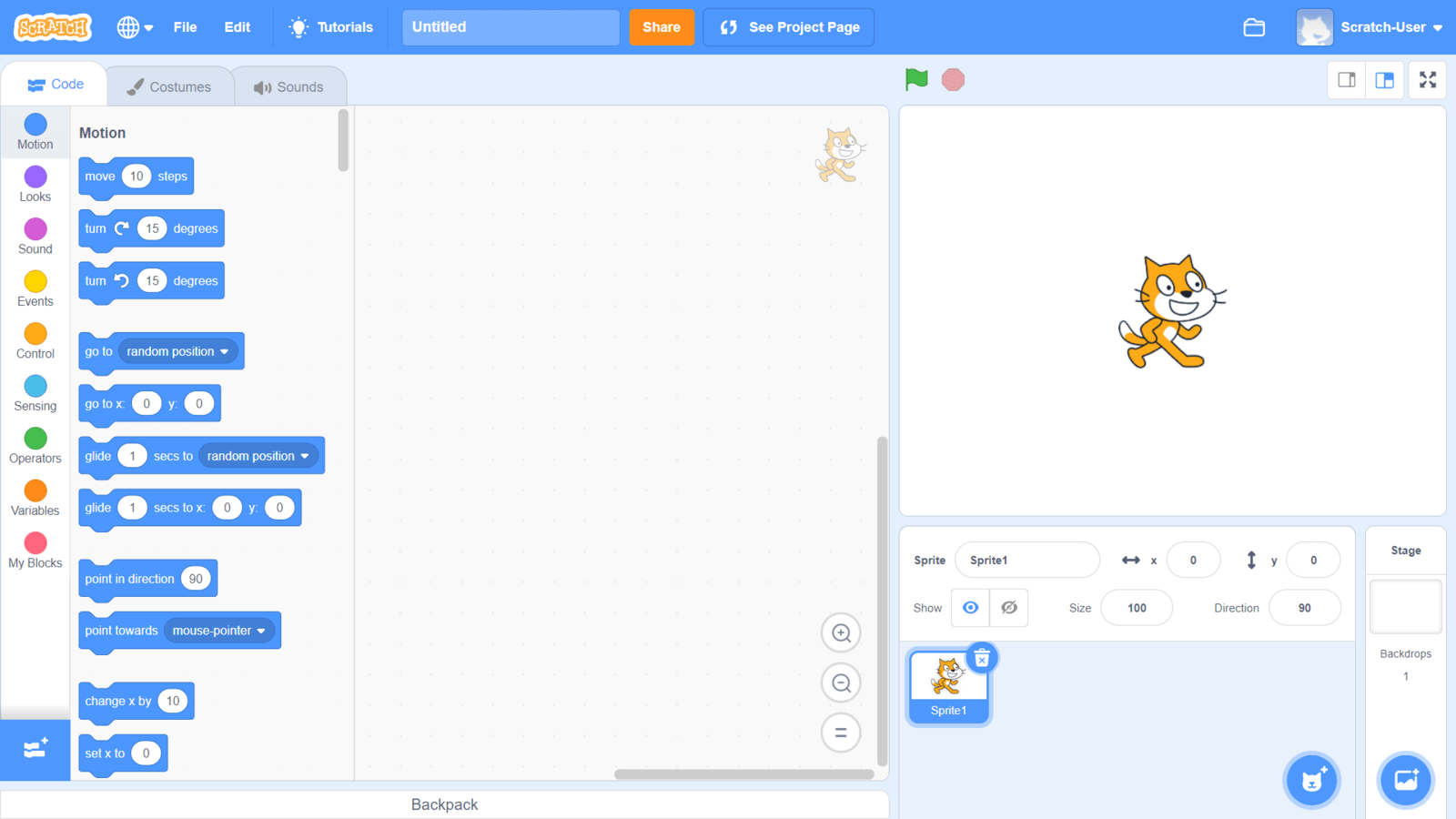
**7-й модуль**

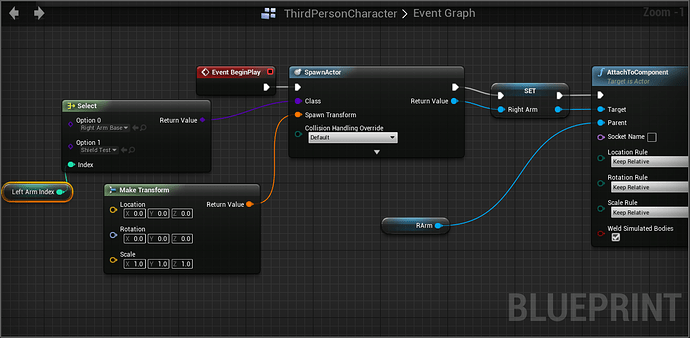
**“Codeblock”**

*Формат работы:* в группах (1-4 человека).

*Задача*: реализовать свой собственный аналог Codeblock (интерпретатор алгоритмов, которые строятся из структурных элементов языка) в форме мобильного приложения.

## Референсы:





Алгоритмическая сложность определяется тем, какими возможностями будет обладать ваше приложение с точки зрения допустимых операций и тем, как они выполняются. Далее будут описаны категории операций, которые так или иначе ожидаются в вашем интерпретаторе. Категории, во-первых, коррелируют со сложностью, во-вторых, идут включительно, т.е. если вы планируете делать последнюю категорию сложности, то все предыдущие должны также быть реализованы и работать с учётом всего дальнейшего усложнения.

Независимо от выбранной категории необходимо реализовать отображение ошибок в алгоритме (подсветка участка с неверным “синтаксисом”, из-за которого дальнейшая интерпретация невозможна).

## Категории:

### “Буквально минимум”

Ваш интерпретатор должен позволять:

* Объявлять целочисленные переменные. В ваши допустимые элементы языка должен быть заложен блок (в прямом смысле, перетаскиваемый по экрану элемент), который позволяет декларировать новую целочисленную переменную и указать ее имя. **Небольшое усложнение:** возможность добавить новые переменные через запятые, а не делать несколько команд для нескольких переменных. Для простоты считаем, что все вновь объявленные переменные по умолчанию равны 0.
* Реализовать базовую логику оператора присваивания. В ваши допустимые элементы языка должен быть заложен блок, которые олицетворяет собой операцию присваивания. В которой вы указываете переменную, к которой применяется операция присваивания, а затем указываете арифметическую операцию.
* Обрабатывать базовые арифметические операции. Ваш интерпретатор должен позволять создавать арифметические выражения, в которые могут входить целочисленные константы, переменные, стандартные операции для работы с целочисленными значениями (сложение, вычитание, умножение, деление целой части, остаток от деления). ДОПУСТИМОЕ, но не поощряемое упрощение: игнорировать арифметический порядок выполнения операций (только если вы не претендуете ни на одну другую категорию).
* Расширять и обрабатывать скобочные последовательности в арифметических операциях, изменяя порядок выполнения операций.
* Оператор If. Допустимый вариант: вы создаете элемент языка, который выполняет функцию логического оператора if, в основу которого можно указать какую-то из операций сравнения (>,<,=,!=,>=,<=) и два арифметических выражения, что с чем сравнивается. Далее создается подблок, куда можно указать набор алгоритмических команд, которые будут выполняться, если логическое условие верно.
* Элемент “на подумать”. Begin-End или его аналоги, в зависимости от структуры хранения и описания ваших алгоритмов. Пара Begin/end может упростить вам работу с вложенными командами. А может не упростить… Намёк. Если ваша логика работы с вложенными блоками не требует Begin/End, то не делайте их.

### “Я могу написать сортировку пузырьком”

Ваш интерпретатор должен позволять:

* Осуществлять логику циклов. Можно сделать, реализовав оператор goto и флаги, или специализированный элемент языка для циклов. Логики While вполне себе достаточно. **Вариант усложнения**: сделать полный аналог for из С++, с возможностью указать отдельные команды, которые выполняются один раз до начала цикла, логическое условие и команду, которая выполняется при новой итерации цикла.
* Расширение логического оператора за счет If-Else. Думаю, комментарии излишни.
* Возможность объявлять и работать со статическими массивами. Объявление массивов с указанием имени и количества элементов массива. Имплементация логики индекса элементов массива. Работа с элементами массива.

В **среднем** ожидается демонстрация работы приложения на имплементированной сортировке пузырьком.

### “HARD”

Следующая категория включает в себя возможные усложнения вашего приложения… На эту категорию не распространяется необходимость реализовывать все ее пункты:

Алгоритмические усложнения интерпретатора

* Возможность объявлять и работать с функциями.
* Использование сложных вложенных арифметических выражений в логических выражениях, и в индексах массивов (если сделан предыдущий блок, то и с учетом вызванных функций).
* Работа со строками и с символьными типами данных.
* Реализация Логических операторов через boolean.
* Вещественный тип переменных.
* Конвертация типов переменных (если в вашем языке есть несколько разных типов переменных).
* Организация структуры вашего кода не через прямое указание команд и их составных частей, а через организацию потоков данных (аналог Blueprint).
* Возможность объявлять и работать со структурами данных.
* Возможность сохранять и работать с файлами самих алгоритмов.
* Любой элемент, которые есть в более “общественно признанных” языках программирования, который делает возможности вашего языка более мощными (Матрицы, структуры данных, ООП, лямбды…)
* Режим отладки и пошагового исполнения (Debug Mode).

**Требования по мобильной части:**

1. Разрешается использовать любые варианты создания мобильного приложения для iOS или Android (нативная разработка, Flutter, React Native и прочие… но! адекватная помощь будет только по нативным инструментам).
2. Можно использовать любые сторонние библиотеки для реализации элементов интерфейса и взаимодействия с пользователем. Использовать библиотеки для основных алгоритмов – нельзя.
3. Будет оцениваться корректность работы с жизненными циклами (приложения и компонентов).
4. Будет оцениваться дизайн и UX. Приложение должно быть красивым и удобным для использования.

Сохраняется требование еженедельного показа прогресса и обязательность наличия кода в Git.

**ВНИМАНИЕ**: преобразование последовательности визуальных блоков в команды с последующим запуском алгоритма с помощью сторонних библиотек, JS-интерпретаторов и прочих хитростей не является решением поставленной задачи и не принимается.