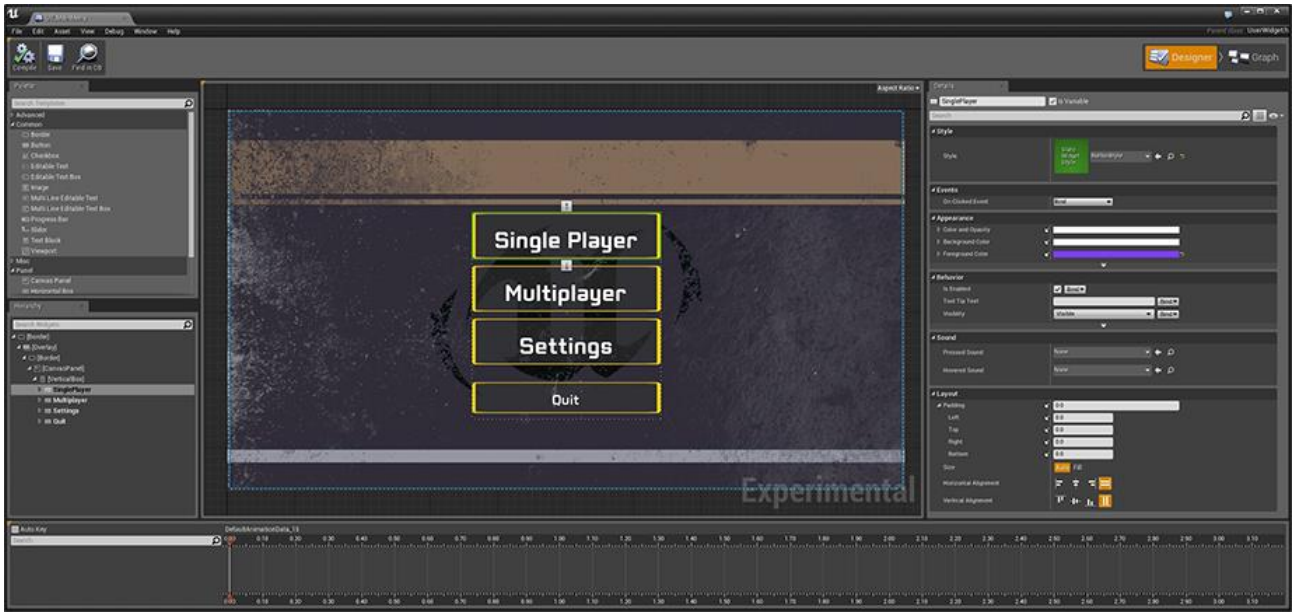
Туториал по Unreal Engine. Часть 4: UI



Разработчики видеоигр используют графику и текст для отображения необходимой информации, например, здоровья или очков. Это называется интерфейсом пользователя (user interface, UI).

UI в Unreal Engine 4 создаётся с помощью Unreal Motion Graphics (UMG). UMG позволяет удобно выстраивать UI, перетаскивая элементы UI, такие как кнопки и текстовые метки.

В этой части туториала вы научитесь следующему:

- Создавать HUD-дисплей, на котором отображается счётчик и таймер
- Отображать HUD на экране
- Обновлять счётчик и таймер, чтобы отображать значения переменных

Приступаем к работе

Скачайте https://koenig-media.raywenderlich.com/uploads/2017/08/GeometryCatcherStarter.zip и распакуйте её. Перейдите в папку проекта и откройте *GeometryCatcher.uproject*.

Примечание: если откроется окно, сообщающее, что проект создан в более ранней версии Unreal editor, то всё в порядке (движок часто обновляется). Можно или выбрать опцию создания копии, или опцию преобразования самого проекта.

Нажмите на *Play*, чтобы начать управлять белым кубом и попробуйте ловить падающие фигуры. Куб можно перемещать горизонтально с помощью мыши. Через десять секунд фигуры перестают создаваться.

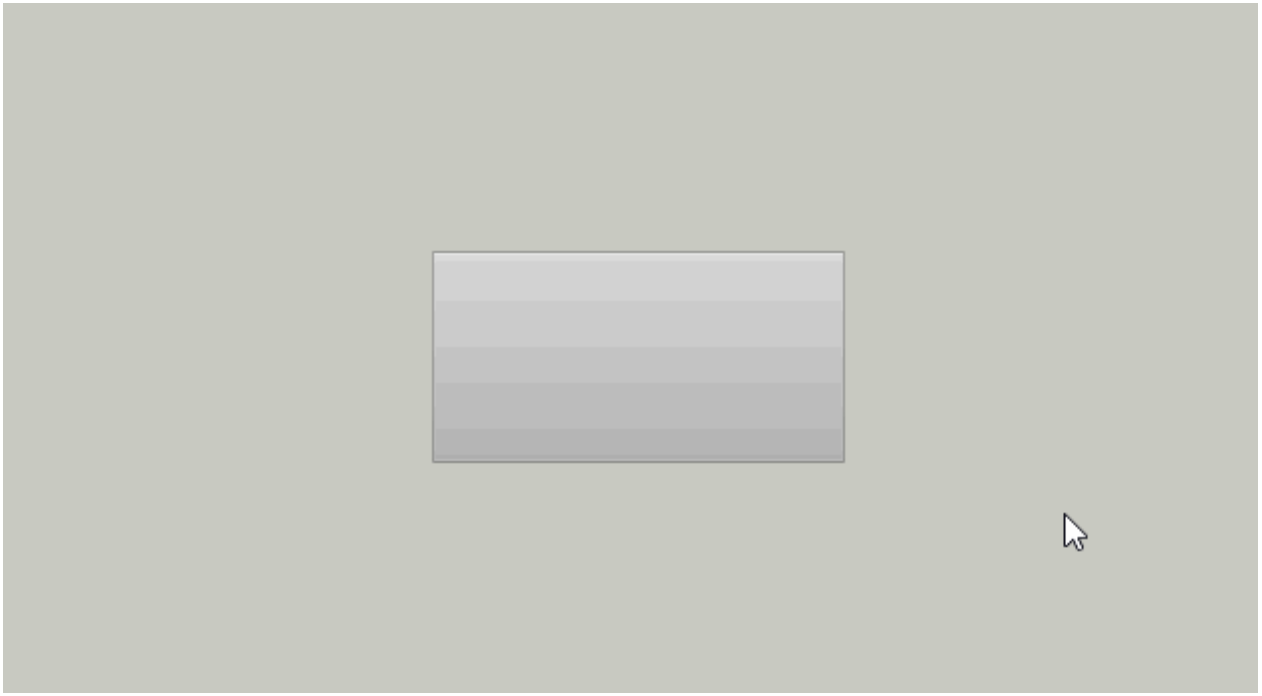
Первое, что нужно сделать — создать HUD-дисплей, отображающий две вещи:

- Счётчик, отслеживающий количество собранных игроком фигур
- Таймер, отображающий количество секунд, оставшихся до завершения создания фигур

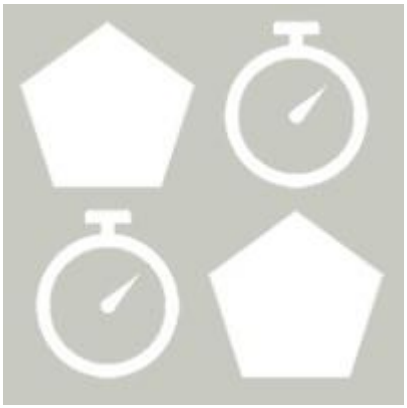
Чтобы создать всё это, нам необходимы *виджеты*.

О виджетах

Виджет (widget) — это элемент UI, предоставляющий UI визуальные функции. Например, виджет Button предоставляет объект, который пользователь может видеть и нажимать на него.

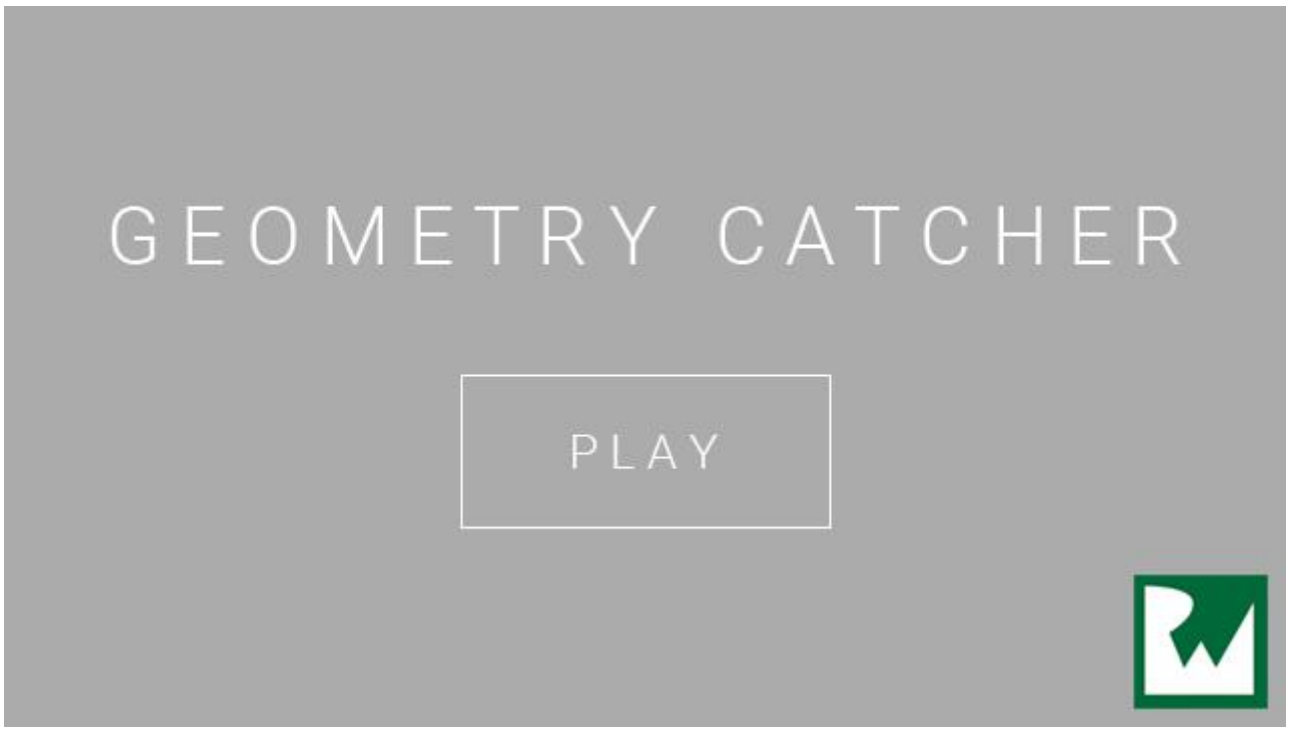


Сам виджет необязательно должен быть видимым. Например, виджет Grid Panel равномерно разделяет своё пространство между его содержимым. Пользователь не может увидеть Grid Panel, но видит его воздействие.



Кроме того, виджеты могут содержать другие виджеты. Вот пример виджета, содержащего виджет Text (метка Name) и виджет Text Box:

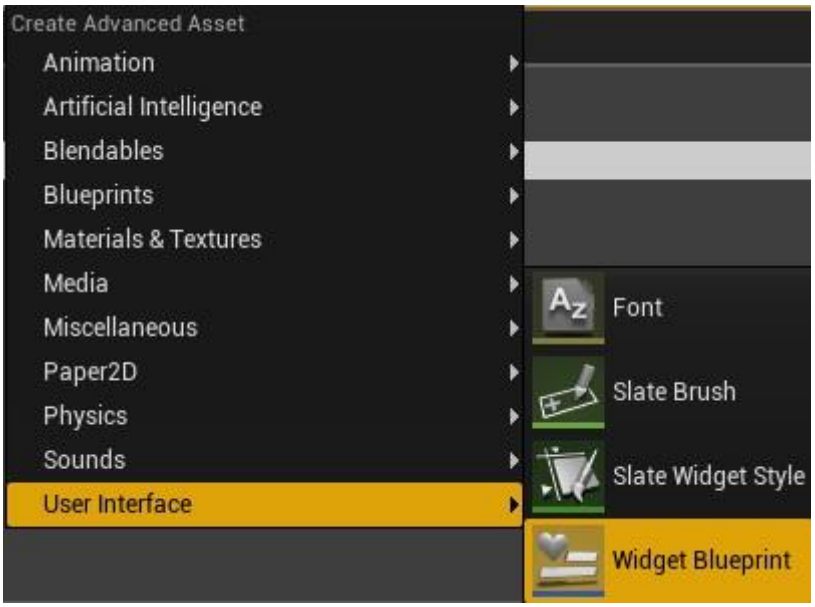
Можно даже создать виджет, являющийся целым интерфейсом, например, экраном меню. Ниже представлен пример виджета, созданного так, чтобы он выглядел как начальный экран игры. Все элементы UI тоже являются виджетами и содержатся внутри виджета начального экрана.



Итак, мы узнали, что такое виджеты. Теперь можно создать виджет для HUD.

Создание виджета

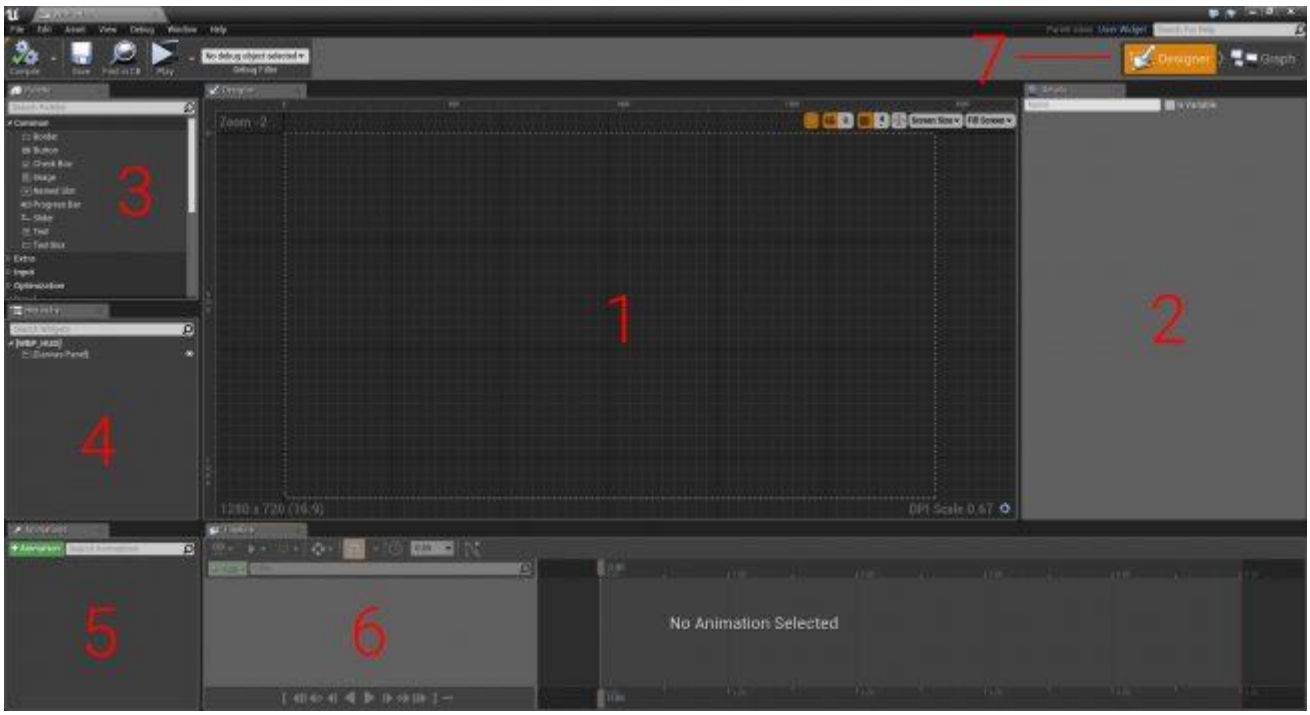
Перейдите в Content Browser и найдите папку *UI*. Нажмите на кнопку *Add New* и выберите *User Interface\Widget Blueprint*. Переименуйте новый ассет в *WBP_HUD*.



Дважды нажмите на *WBP_HUD*, чтобы открыть его в UMG UI Designer.

UMG UI Designer

UMG UI Designer состоит из семи основных элементов:

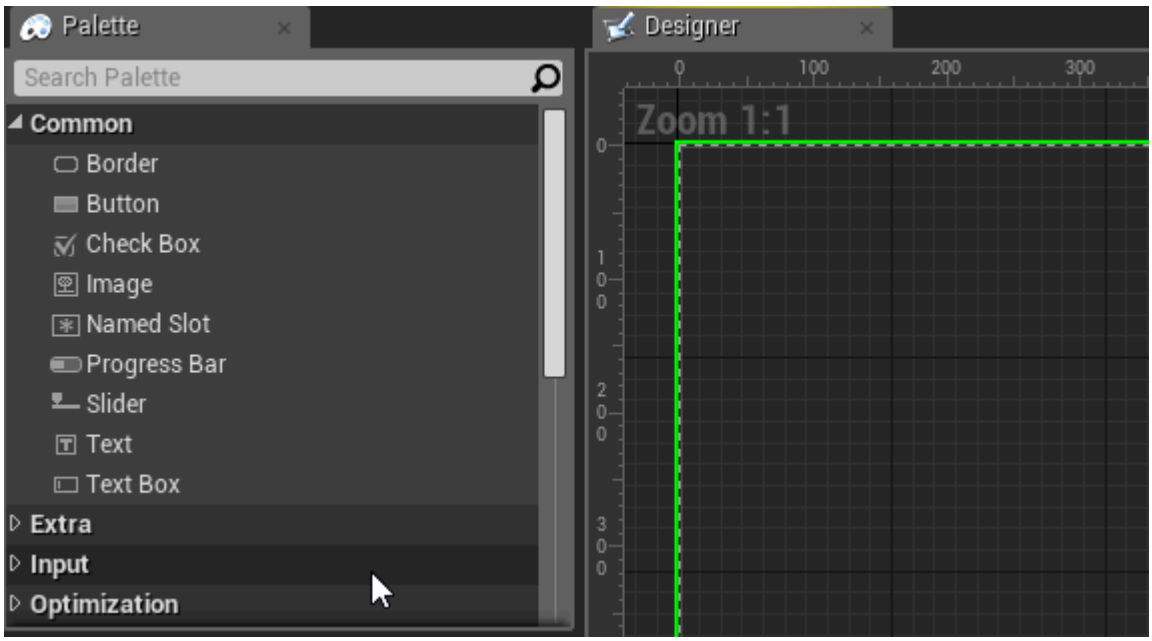


1. *Designer:* в этой области представлено визуальное отображение виджета. Перемещаться по ней можно *зажав правую клавишу мыши* и *двигая мышью*. Масштабирование выполняется *прокруткой колёсика мыши*.
2. *Details:* здесь отображаются свойства выбранного виджета
3. *Palette:* список всех виджетов, которые можно использовать. Все созданные пользователем виджеты тоже появляются здесь.
4. *Hierarchy:* список всех уже используемых виджетов
5. *Animations:* некоторые свойства виджетов могут иметь анимацию, например, расположение и размер. В этой панели перечислены все анимации.
6. *Timeline:* при выборе анимации на этой панели показываются анимированные свойства и ключевые кадры
7. *Editor Mode:* здесь можно переключаться между режимами Designer и Graph. Режим Graph почти аналогичен Event Graph у Blueprint.

Создание виджета Text

Виджеты Text отлично подходят для отображения числовой информации, например, счётчика и таймера.

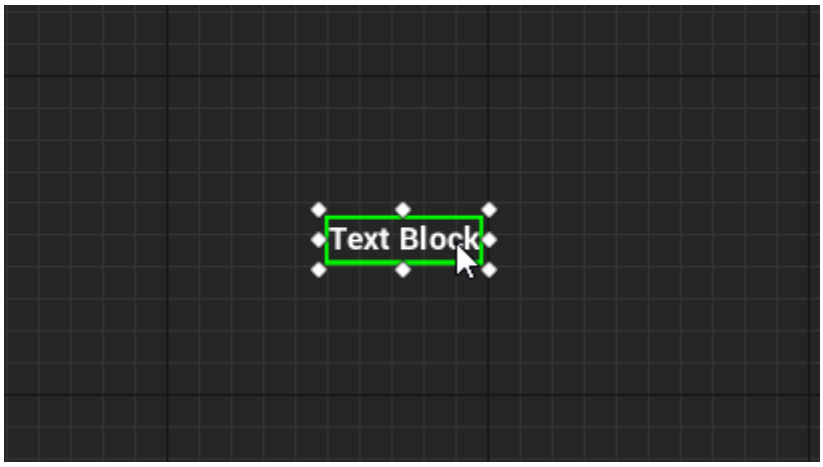
Перейдите в панель Palette и найдите виджет *Text*. Добавьте виджет, *перетащив его мышью* в панель Designer.



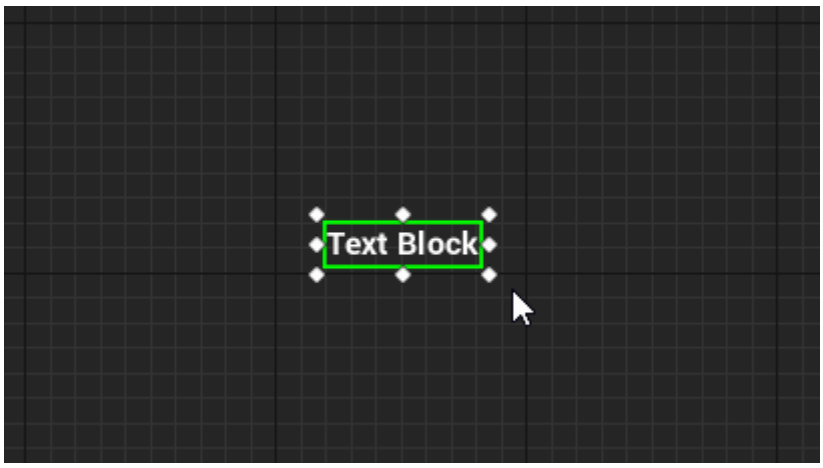
Содержание текста нас пока не интересует, мы изменим его позже.

Переименуйте виджет в *CounterText*. Это можно сделать, выбрав виджет *Text* и перейдя в панель Details. Введите *CounterText* в *текстовое поле* в верхней части.

Виджеты можно двигать, *перетаскивая их левой клавишей мыши* в панели Designer.

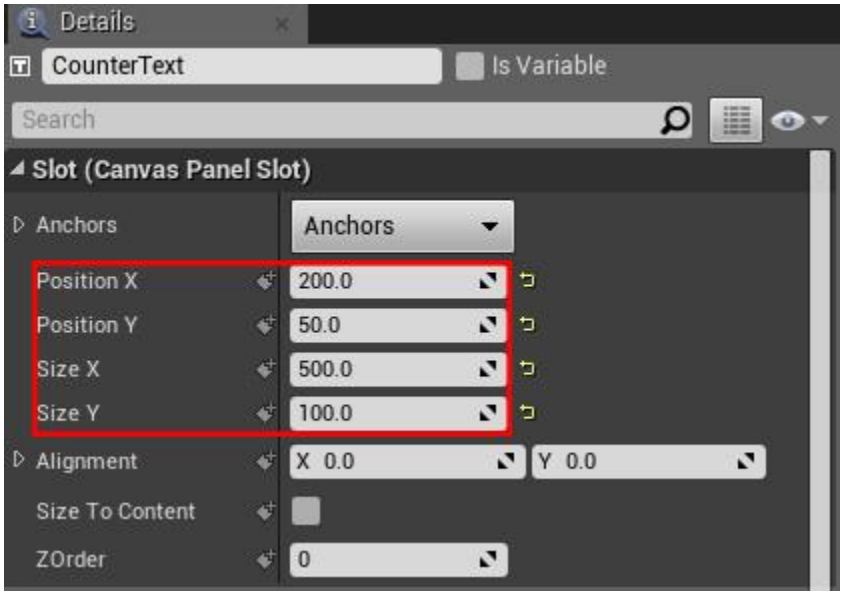


Также можно изменять размер виджетов, *перетаскивая левой клавишей мыши границы*. Изменение размера позволяет задать границы виджета. Unreal не будет ничего рендерить за пределами границ.

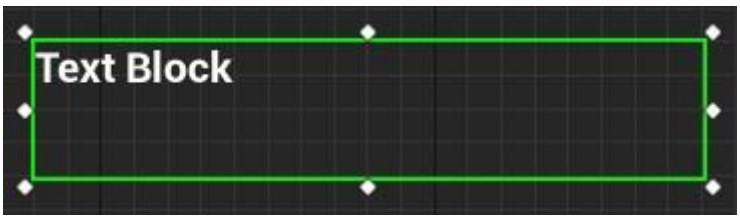


Или же можно задать положение и размер изменением значений в панели Details. Задайте следующие свойства и значения для *CounterText*:

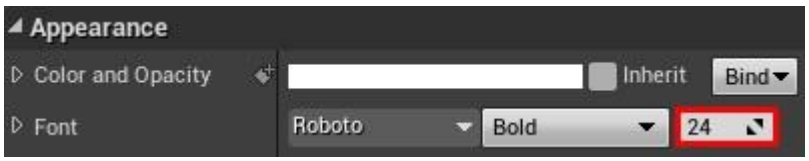
- *Position X:* 200
- *Position Y:* 50
- *Size X:* 500
- *Size Y:* 100



На данный момент текст занимает только небольшую часть поля.



Можно увеличить размер шрифта, перейдя в панель Details и к разделу *Appearance*. Справа от свойства *Font* есть текстовое поле для задания размера шрифта.



Введите размер *68*.

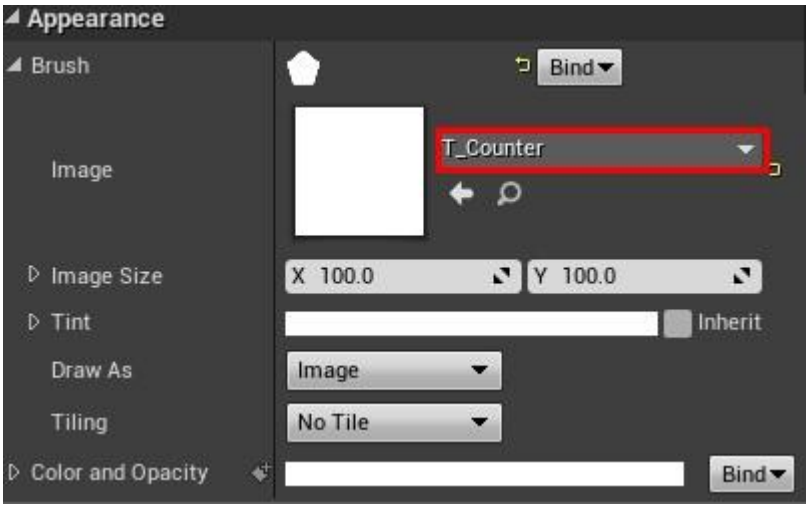
Давайте сделаем счётчик красивее, добавив к нему значок.

Создание виджета Image

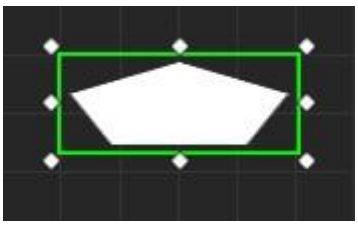
Виджеты Image — это простой способ отображения графики в UI, например, значков.

Создайте виджет *Image* и назовите его *CounterIcon*. Задайте *Position X* значение *75*, а *Position Y* — значение *50*. Виджет разместится рядом с *CounterText*.

Чтобы задать изображение, перейдите в панель Details и зайдите в раздел *Appearance*. Разверните свойство *Brush*, а затем нажмите на *раскрывающемся списке* рядом с *Image*. Выберите *T_Counter*.

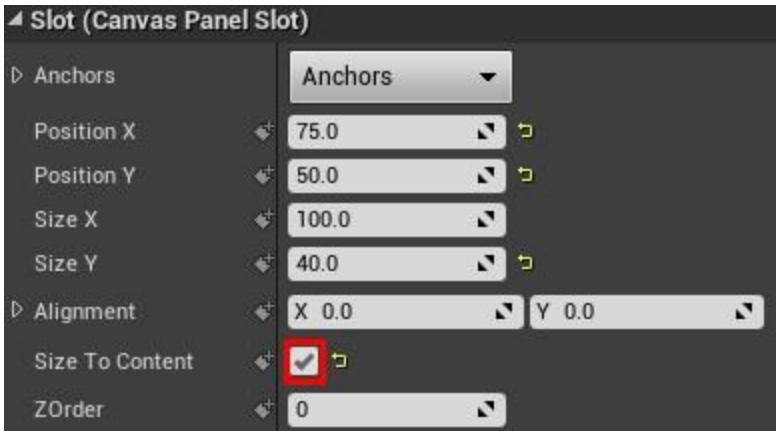


Изображение будет выглядеть растянутым, потому что размер виджета отличается от размера изображения.



Вместо изменения размера виджета мы можем использовать опцию *Size To Content*. Эта опция автоматически изменяет размер виджета под размер его содержимого.

Находясь в панели Details, перейдите в раздел *Slot (Canvas Panel Slot)*. Поставьте *флажок* рядом с *Size To Content*.



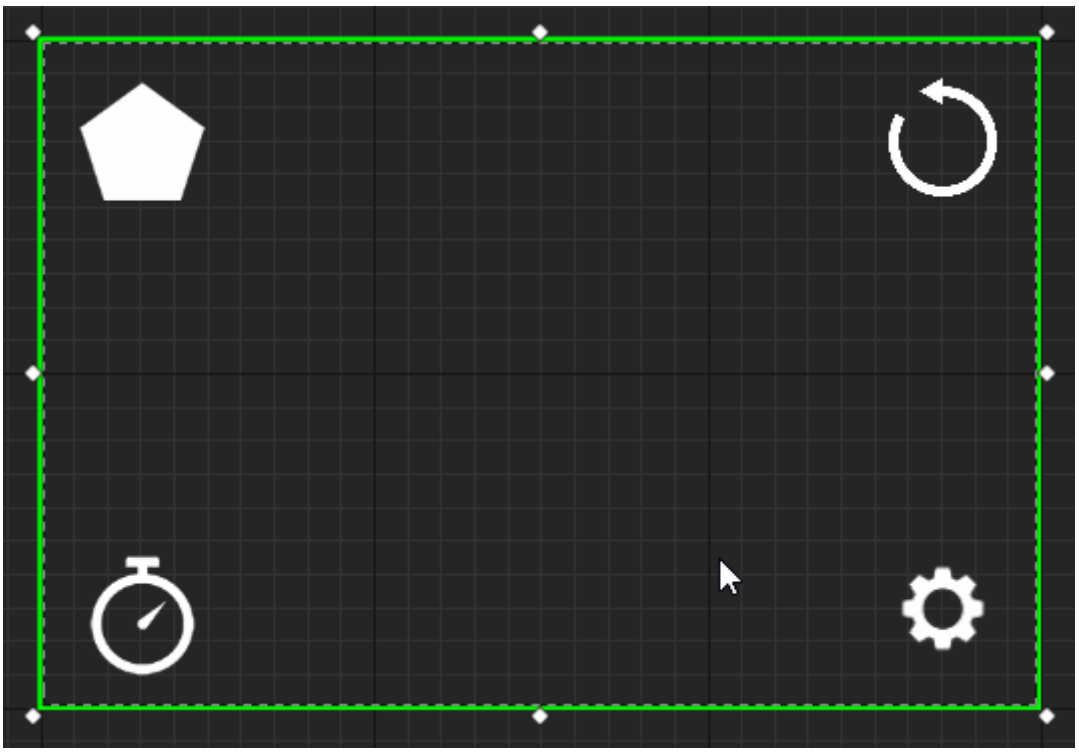
Виджет сам подгонит свой размер под изображение.

Если игра будет запускаться с разными размерами экрана, то UI должен соответствующим образом перемещать виджеты. Чтобы сохранить компоновку UI, нужно использовать *привязки*.

Привязки

Точка привязки задаёт место, относительно которого определяется положение виджета. По умолчанию виджеты привязаны к верхнему левому углу своего родительского элемента. Поэтому когда мы задаём положение виджета, мы на самом деле указывает положение относительно этой точки привязки.

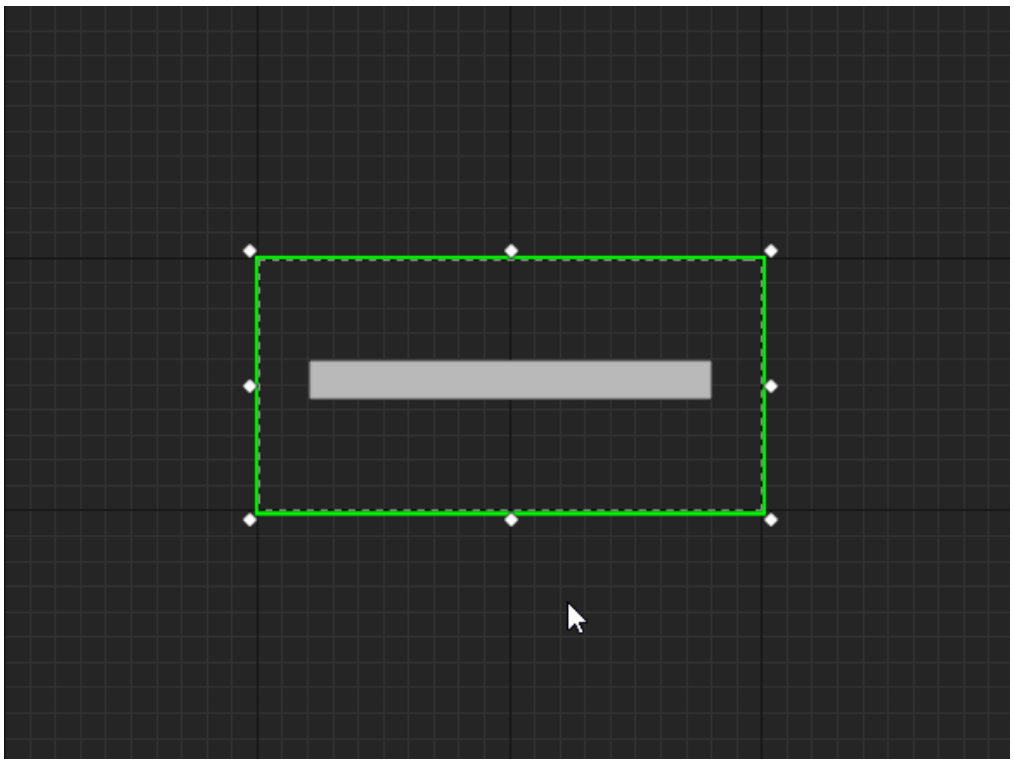
В примере ниже каждое изображение привязано к одной точке (к ближайшему углу).



Заметьте, что каждое изображение сохраняет положение относительно своей привязки. Благодаря привязкам UI будет иметь одинаковое расположение при разных размерах экрана.

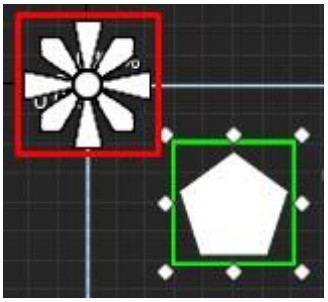
Также можно использовать привязки для автоматического изменения размера виджетов. В случае привязки к двум или более точкам виджет будет менять свой размер для сохранения относительного размера.

В примере ниже индикатор привязан к верхнему левому и верхнему правому углам.



Вертикально индикатор перемещается, но не меняет размера, потому что на оси Y есть только одна привязка (сверху). Однако горизонтально индикатор меняет размер, потому что имеет две точки привязки по оси X.

Anchor Medallion отображает расположение привязки. Он появляется при выборе виджета.



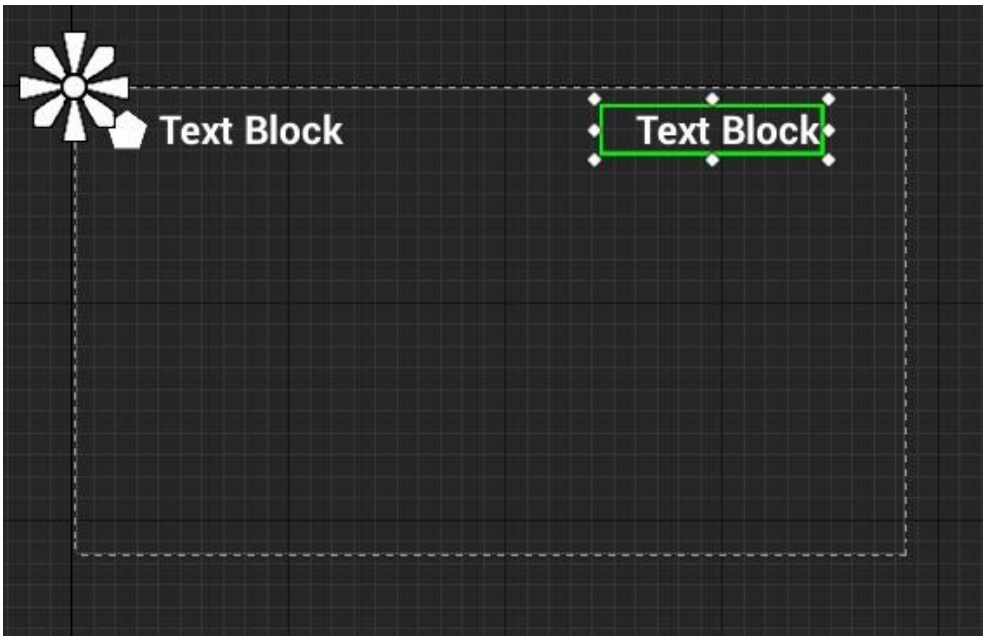
Привязки *CounterText* и *CounterIcon* уже находятся в нужном месте, поэтому задавать их не нужно.

Теперь мы создадим ещё по одному виджету Text и Image для таймера. Однако на этот раз мы поместим их справа.

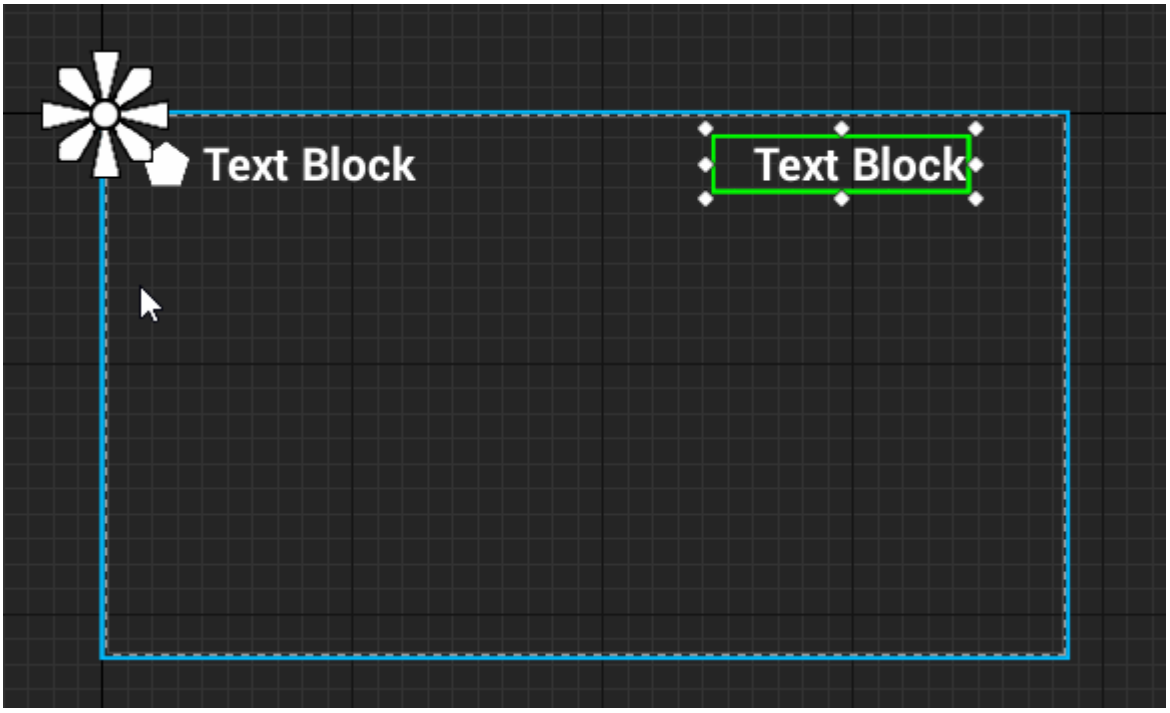
Создание таймера

Создайте виджет *Text* и назовите его *TimerText*. Задайте следующие свойства:

- *Position X:* 1225
- *Position Y:* 50
- *Size X:* 500
- *Size Y:* 100
- *Font Size:* 68
- *Justification:* Align Text Right (это выровняет текст по правой стороне виджета)



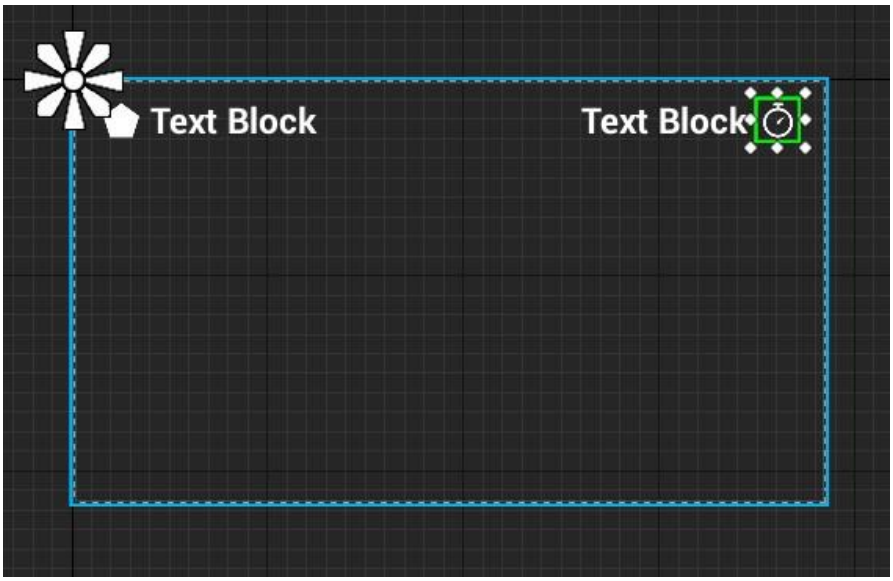
Теперь мы хотим задать привязку в правом верхнем углу. Это можно сделать, *перетащив мышью круг* в *Anchor Medallion*. Переместите *Anchor Medallion* в *правый верхний угол*.



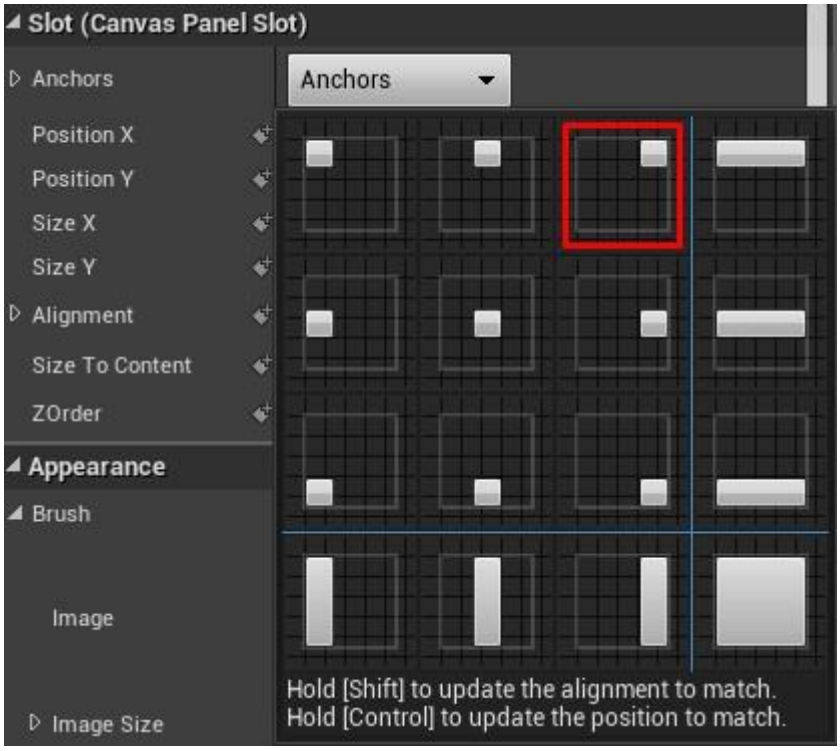
Заметьте, как обновляется положение относительно привязки.

Создайте виджет *Image* и назовите его *TimerIcon*. Задайте следующие свойства:

- *Position X:* 1750
- *Position Y:* 50
- *Size To Content:* Checked
- *Brush\Image:* T_Timer

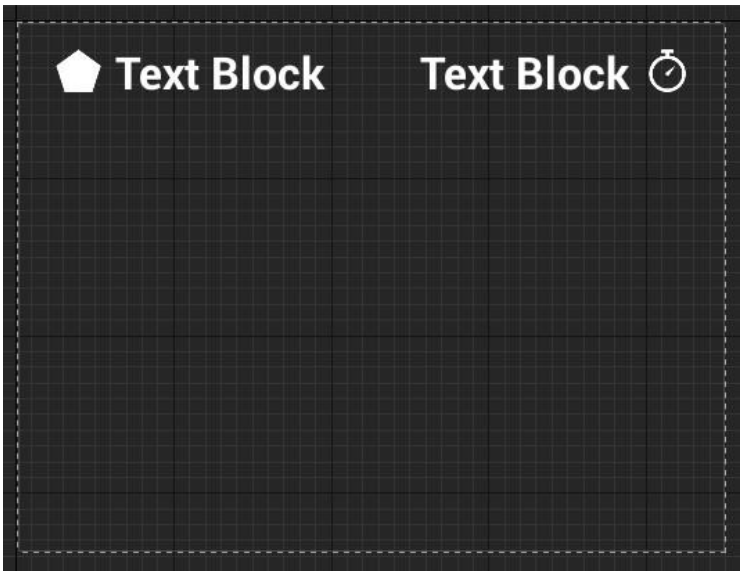


Вместо задания привязки с помощью Anchor Medallion, можно воспользоваться предустановленными значениями. Перейдите в панель Details и нажмите на *раскрывающийся список* рядом с *Anchors*, чтобы открыть предустановленные значения. Выберите *третью* схему (с квадратом в правом верхнем углу).



Создание схемы UI на этом завершено. Можно убедиться в том, что привязки работают, эмулируя различные размеры экрана. Перейдите в панель Designer и нажмите на раскрывающийся список *Screen Size*.

Выбор опции изменит размер *WBP_HUD* для соответствия опции. Ниже показано, как HUD будет выглядеть на iPad Air. Заметьте, что виджеты стали ближе друг к другу.



В следующем разделе мы узнаем, как отображать виджет *WBP_HUD*.

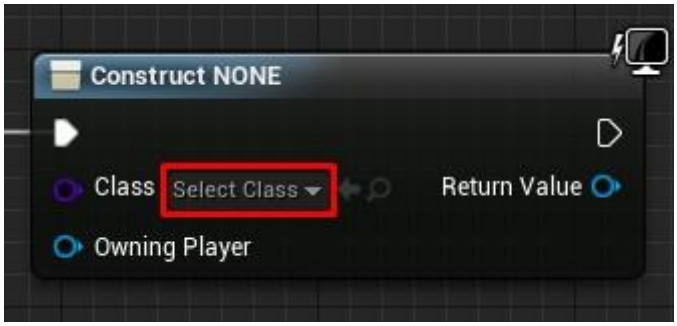
Отображение HUD

Нажмите на *Compile*, а затем вернитесь в основной редактор. Перейдите в папку *Blueprints* и *дважды щёлкните* на *BP_GameManager*, чтобы открыть его.

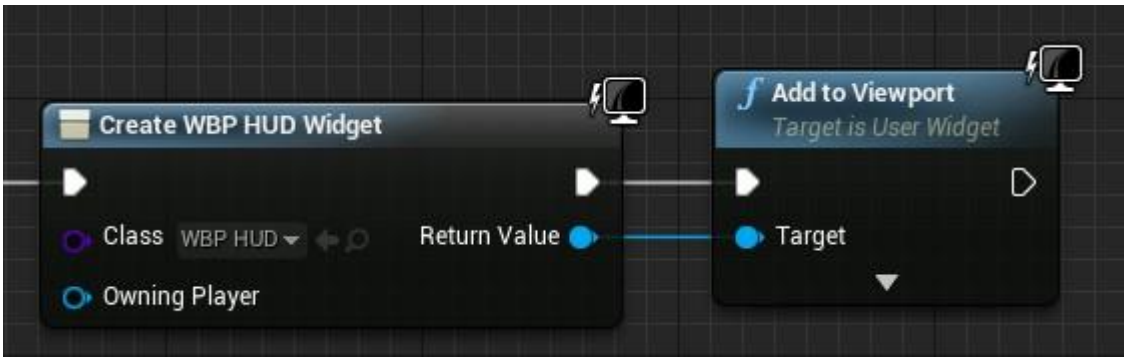
HUD должен становиться видимым после запуска игры. Для этого можно использовать нод *Event BeginPlay*.

Найдите нод *Event BeginPlay* и добавьте нод *Create Widget* в конец цепочки нодов. Этот нод создаёт экземпляр указанного виджета.

Нажмите на *раскрывающийся список* рядом с *Class* и выберите *WBP_HUD*.



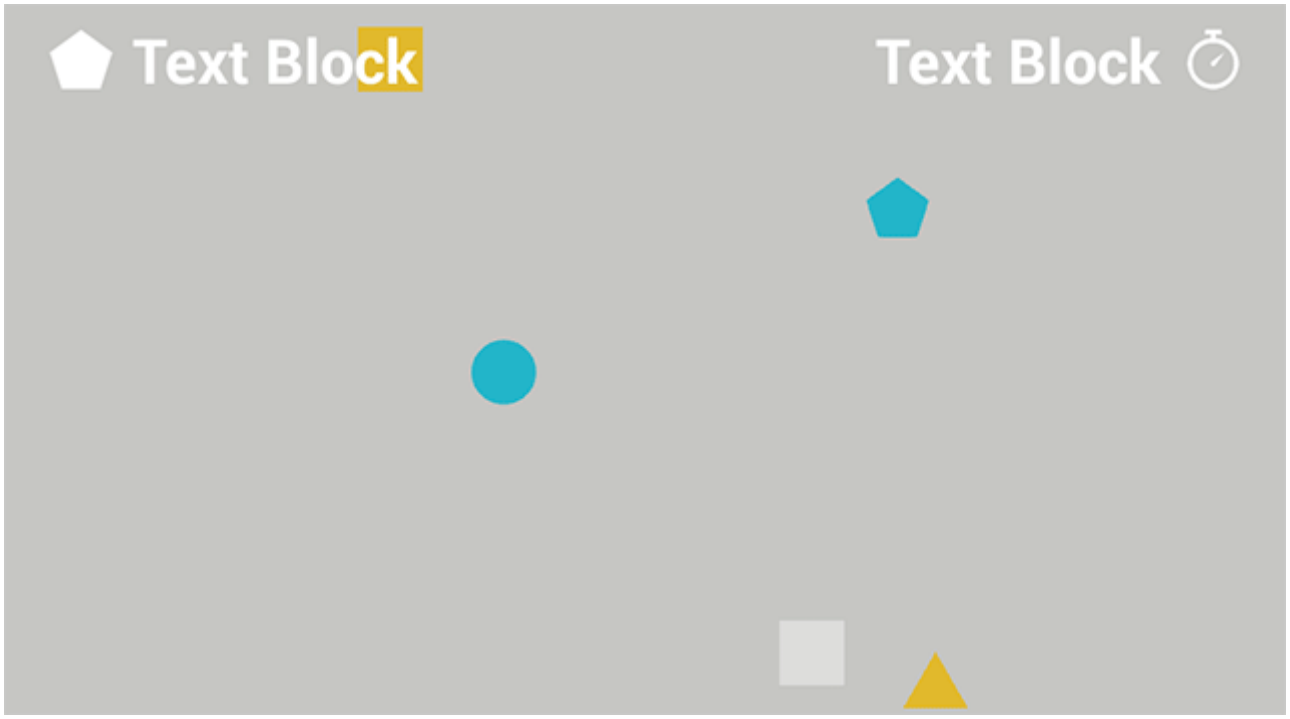
Чтобы отобразить HUD, необходимо использовать нод *Add to Viewport*. *Перетащите левой клавишей мыши* контакт *Return Value* нода *Create Widget*. Отпустите *левую клавишу мыши* в пустом пространстве, чтобы открыть контекстное меню. Добавьте нод *Add to Viewport*.



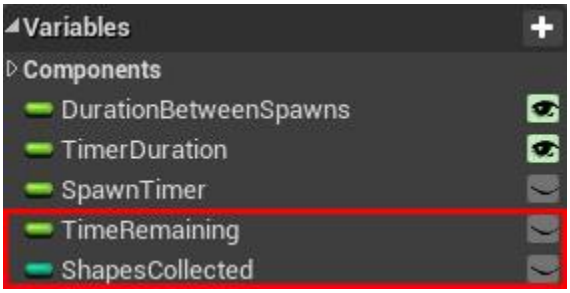
Давайте разберёмся с порядком событий:

1. Когда Unreal спаунит *BP_GameManager*, выполняются функции *Restart* и *SetUpCamera*. Эти функции настраивают несколько переменных и камеру. Если вы не знаете, что такое функция, то не волнуйтесь, скоро мы их рассмотрим.
2. Нод *Create Widget* создаёт экземпляр *WBP_HUD*
3. Нод *Add to Viewport* отображает *WBP_HUD*

Нажмите на *Compile* и вернитесь в основной редактор. Нажмите на *Play*, чтобы запустить игру с новым HUD.



Чтобы отобразить значения счётчика и таймера, нам нужны переменные для хранения этой информации. Эти переменные можно найти в *BP_GameManager*.



Чтобы использовать эти переменные. нам нужен способ получения доступа к *BP_GameManager* из *WBP_HUD*. Для этого можно применить *переменную-ссылку*.

Переменные-ссылки

Хранить ссылки полезно, потому что они позволяют удобно получать доступ к экземплярам.

Представьте, что у вас есть одна коробка, в которой лежит мяч. Если нам нужно найти и изучить мяч, то это будет просто, потому что у нас есть всего одна коробка.

А теперь представьте, что у нас сотня коробок, но мяч есть только в одной. Нам нужно будет проверять каждую коробку, пока мы не найдём коробку с мячом.

Каждый раз, когда нам нужно будет изучить мяч, придётся выполнять эту операцию. Это быстро приведёт к проблемам с производительностью.

Благодаря ссылкам можно отслеживать коробку с мячом. Таким образом, нам не придётся проверять каждую коробку.

Создание переменной-ссылки

Откройте *WBP_HUD* и переключитесь в режим Graph, перейдя в Editor Mode и выбрав *Graph*.

Перейдите на вкладку My Blueprint и создайте новую переменную *GameManager*.

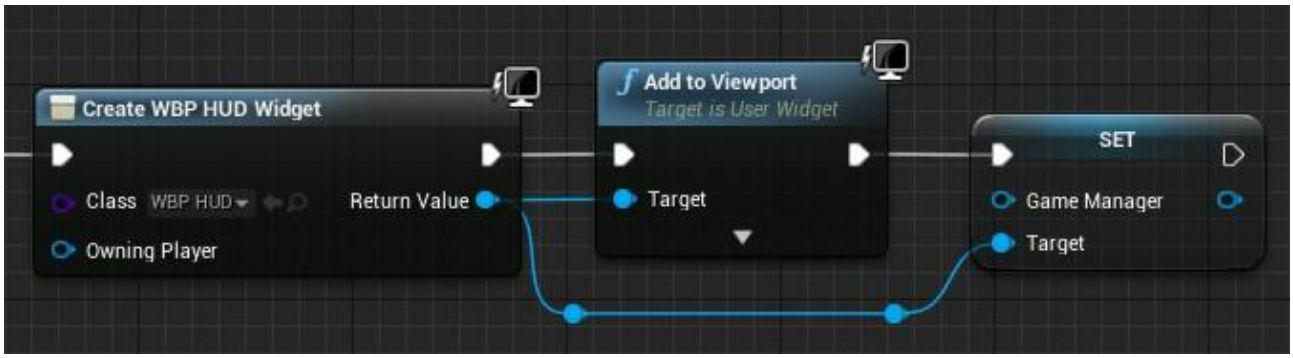
Перейдите в панель Details и нажмите на *раскрывающийся список* рядом с *Variable Type*. Найдите *BP_GameManager* и выберите *BP Game Manager\Object Reference*.

Задание переменной-ссылки

Нажмите на *Compile* и откройте *BP_GameManager*.

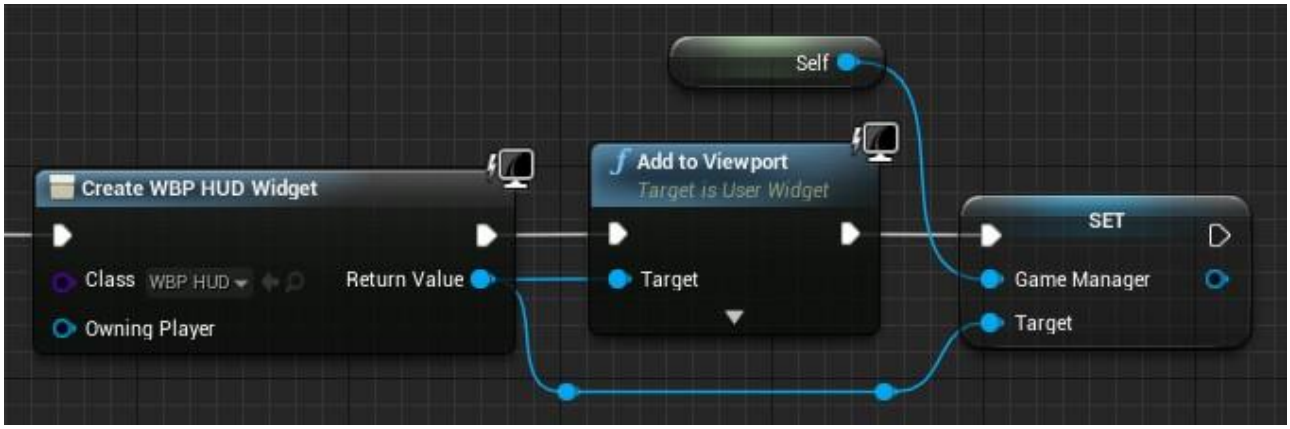
Найдите нод *Create Widget* и *перетащите левой клавишей мыши* контакт *Return Value*. Отпустите *левую клавишу* на пустом пространстве и выберите из меню *Set Game Manager*.

Затем соедините нод *Add to Viewport* с нодом *Set Game Manager*.



Примечание: можно перенаправлять «провода», *дважды щёлкая* на них для создания нода *Reroute*. *Перетащите левой клавишей мыши* нод *Reroute*, чтобы перенаправить «провод».

Затем создайте нод *Self* и соедините его с *левым* контактом нода *Set Game Manager*. Нод *Self* будет указан в списке как *Get a reference to self*.



Теперь, когда *WBP_HUD* уже создан, у нас будет ссылка на *BP_GameManager*.

В следующем разделе мы узнаем, как обновлять виджет с помощью *функций*.

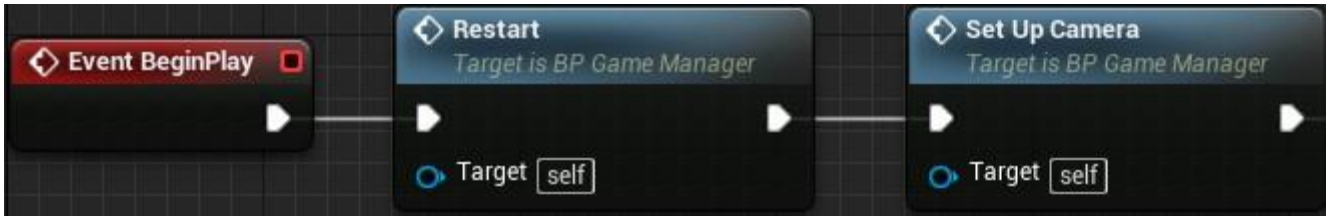
Функции

Функции в Blueprints — это графы, похожие на Event Graph. В отличие от Event Graph функции можно вызывать с помощью нодов. Зачем же это может понадобиться?

Упорядоченность

Одна из причин для использования функций — упорядоченность. Благодаря функциям можно соединить несколько нодов в один.

Посмотрите на раздел *Event BeginPlay* в *BP_GameManager*. Здесь есть две функции: *Restart* и *SetUpCamera*.



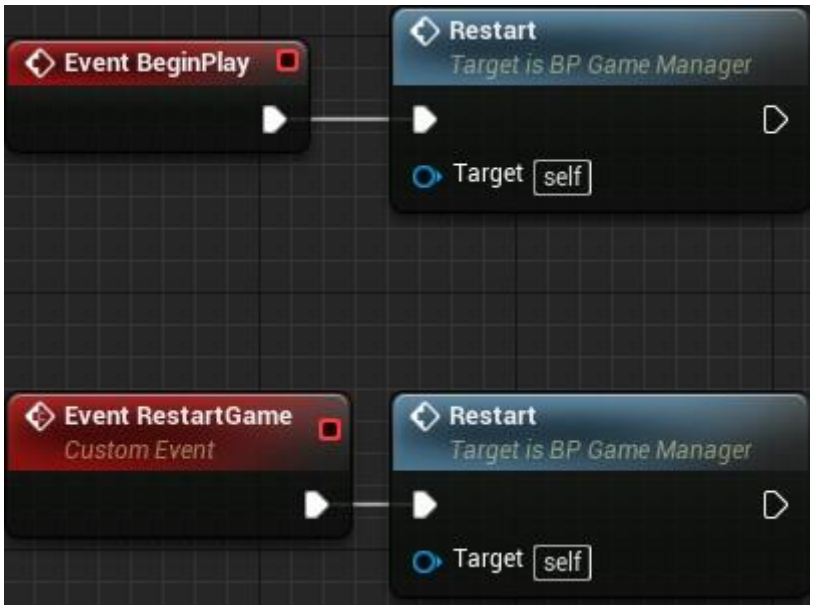
Вот как будет выглядеть этот раздел без функций:

Как вы видите, благодаря функциям он выглядит гораздо чище.

Повторное использование

Ещё одна причина для применения функций — *повторное использование*. Например, если

вам нужно сбросить счётчик и таймер, то это можно запросто сделать с помощью функции *Restart*.



Это позволяет сэкономить время на воссоздание нодов каждый раз, когда вам нужно сбросить эти переменные.

Мы познакомились с функциями и теперь можем использовать их для обновления виджета *CounterText*.

Обновление виджета

При создании виджета также автоматически создаётся и переменная-ссылка на этот виджет. Однако по умолчанию виджеты Text не имеют переменных-ссылок. Это значит, что мы не сможем задавать их свойство Text. К счастью, это легко исправить.

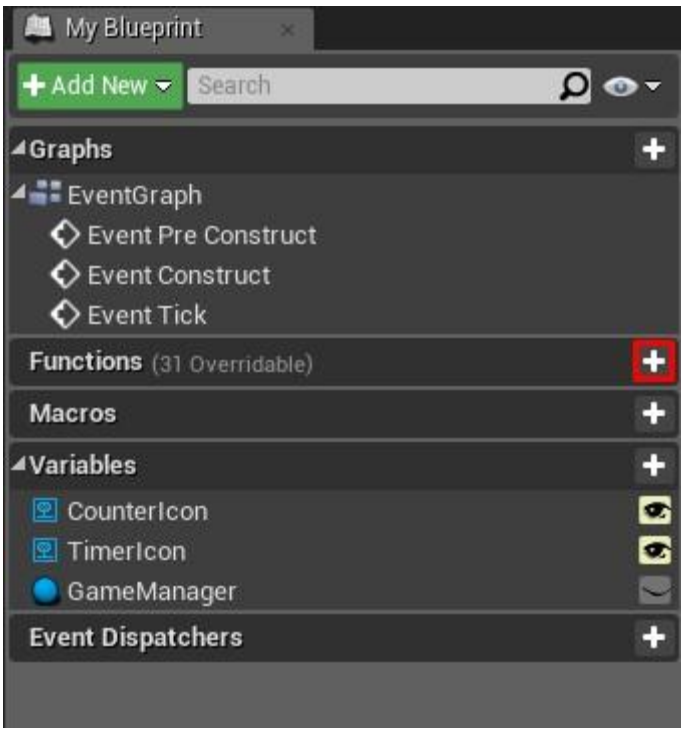
Нажмите на *Compile* и откройте *WBP_HUD*. Переключитесь в режим *Designer*.

Выберите *CounterText*, а затем перейдите в панель Details. Убедитесь, что в самом верху есть флажок *Is Variable*.

Теперь мы сможем обновлять *CounterText*. Следующим шагом будет создание функции для обновления текста.

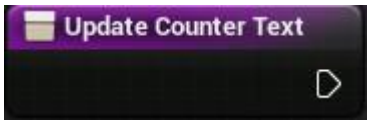
Создание функции обновления

Переключитесь обратно в режим *Graph* и перейдите во вкладку My Blueprint. Нажмите на значок + справа от раздела *Functions*.



При этом будет создана новая функция и вы перейдёте к её графу. Переименуйте функцию в *UpdateCounterText*.

По умолчанию граф будет содержать нод *Entry*. При выполнении функции она будет начинать с этого нода.



Чтобы *CounterText* отображал переменную *ShapesCollected*, нам нужно соединить их.

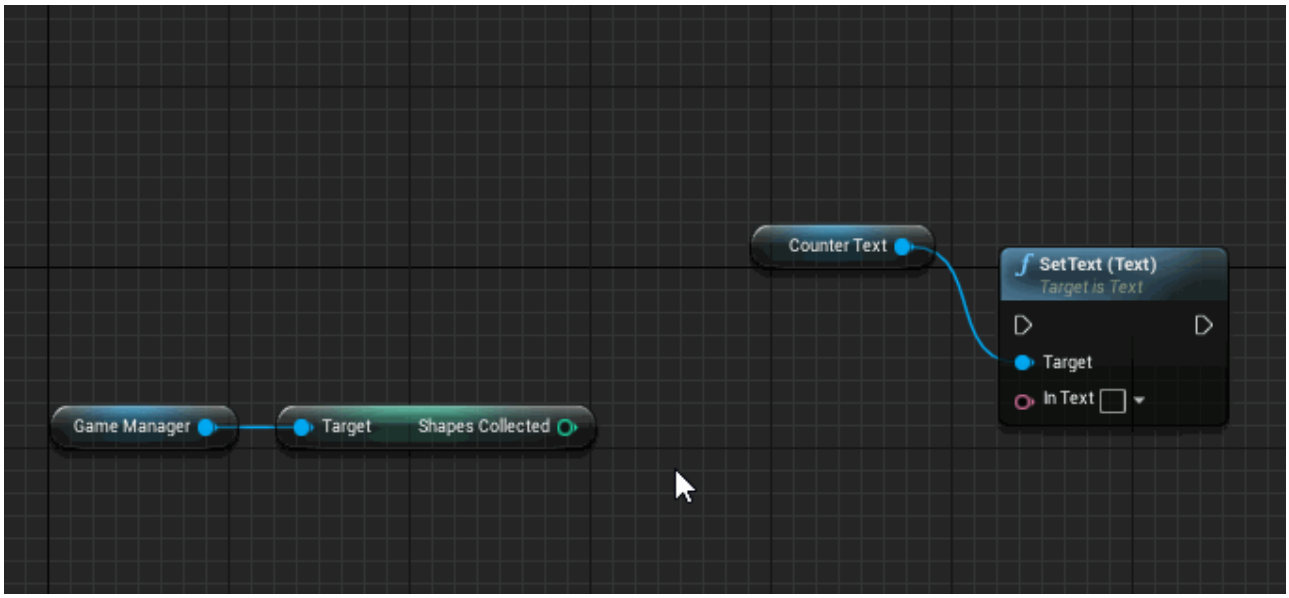
Перетащите в граф переменную *GameManager*. *Перетащите левой клавишей мыши* его контакт и отпустите на пустом пространстве. В меню выберите *Get Shapes Collected*.

Чтобы задать текст, нужно будет использовать нод *SetText (Text)*. Перетащите переменную *CounterText* в граф. *Перетащите левой клавишей мыши* её контакт и отпустите на пустом пространстве. В меню добавьте нод *SetText (Text)*.

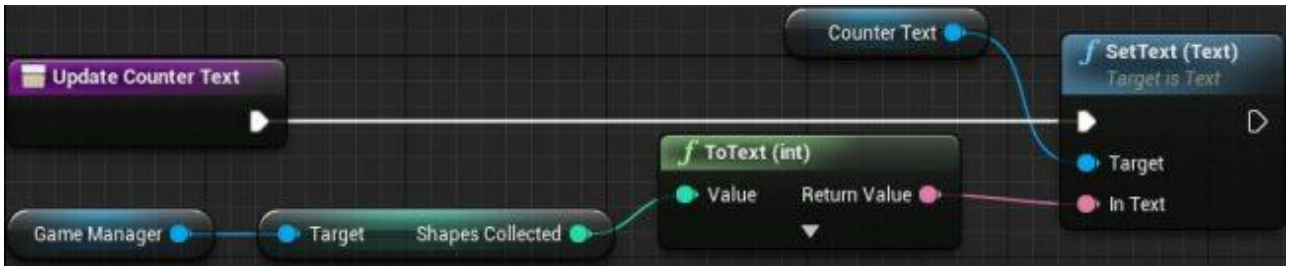
SetText (Text) может получать входные данные типа *Text*. Однако переменная *ShapesCollected* имеет тип *Integer*. К счастью, Unreal автоматически выполняет

преобразование при подключении *Integer* ко входу *Text*.

Соедините переменную *ShapesCollected* с контактом *In Text* нода *Set Text (Text)*. Unreal автоматически создаст нод *ToText (int)*.



Чтобы доделать функцию, соедините нод *Entry* с нодом *Set Text (Text)*.



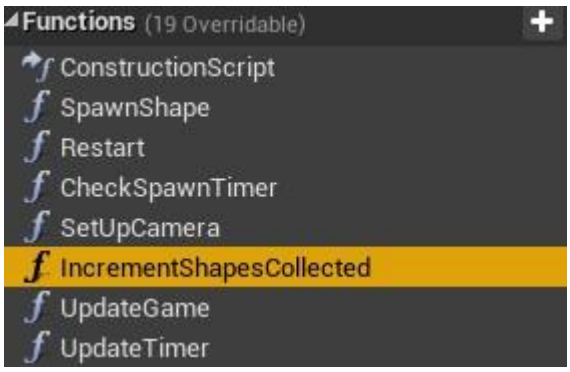
Порядок событий:

1. При вызове *UpdateCounterText* функция получает переменную *ShapesCollected* от *BP_GameManager*
2. Нод *ToText (int)* преобразует значение *ShapesCollected* в тип *Text*
3. *SetText (Text)* задаёт текст *CounterText* из значения *ToText (int)*

Теперь нам нужно научиться вызывать *UpdateCounterText*, когда игрок ловит фигуру.

Вызов функции обновления

Лучше всего вызывать *UpdateCounterText* сразу после того, как игра увеличивает значение *ShapesCollected*. Я создал функцию *IncrementShapesCollected*, которая выполняет инкремент счётчика. Фигуры вызывают эту функцию, когда сталкиваются с игроком.

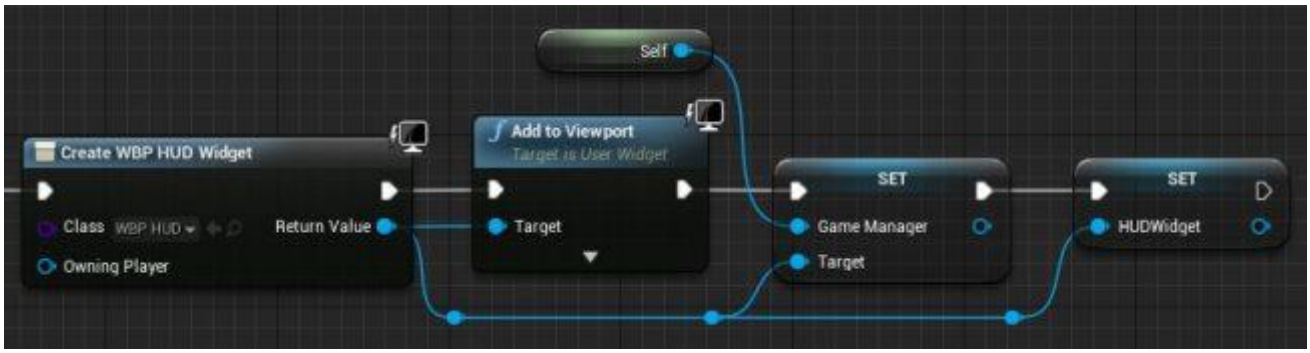


Нажмите на *Compile* и вернитесь в *BP_GameManager*.

Прежде чем вызвать *UpdateCounterText*, нам нужна ссылка на *WBP_HUD*. Попробуйте создать переменную-ссылку самостоятельно!

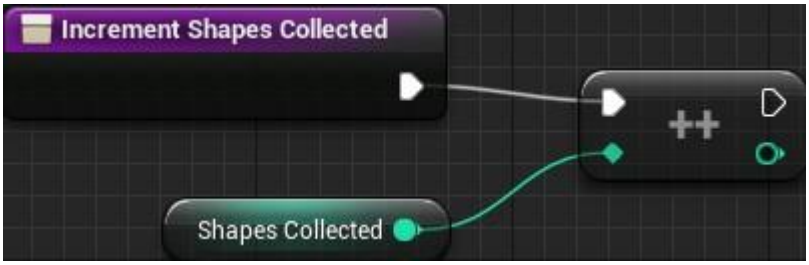
Решение внутри

После создания переменной измените её имя на *HUDWidget*.



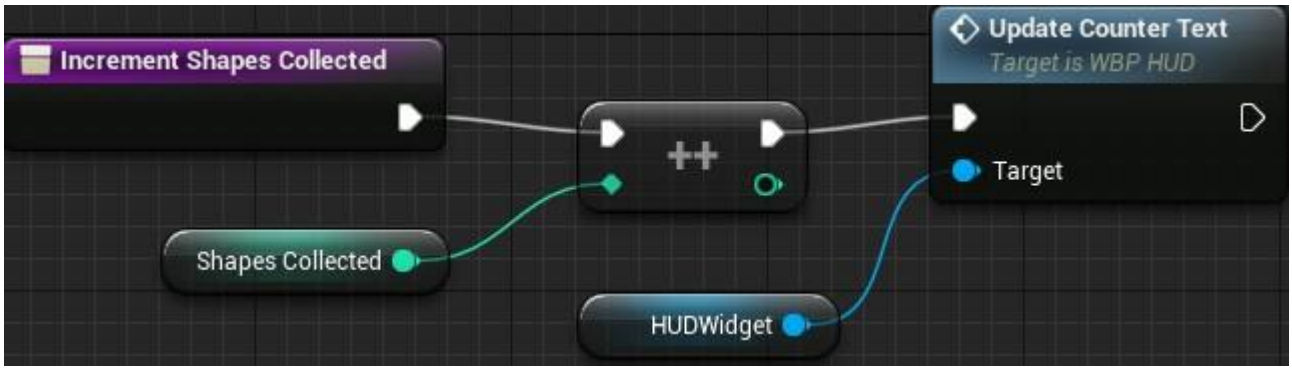
Затем *перетащите правый контакт* нода *Set HUDWidget* и *отпустите* на пустом пространстве. Добавьте нод *UpdateCounterText*. Благодаря этому *CounterText* будет отображать значение *ShapesCollected* при запуске игры.

Затем перейдите в панель My Blueprint и зайдите в раздел *Functions*. *Дважды щёлкните* на *IncrementShapesCollected*, чтобы открыть его граф.



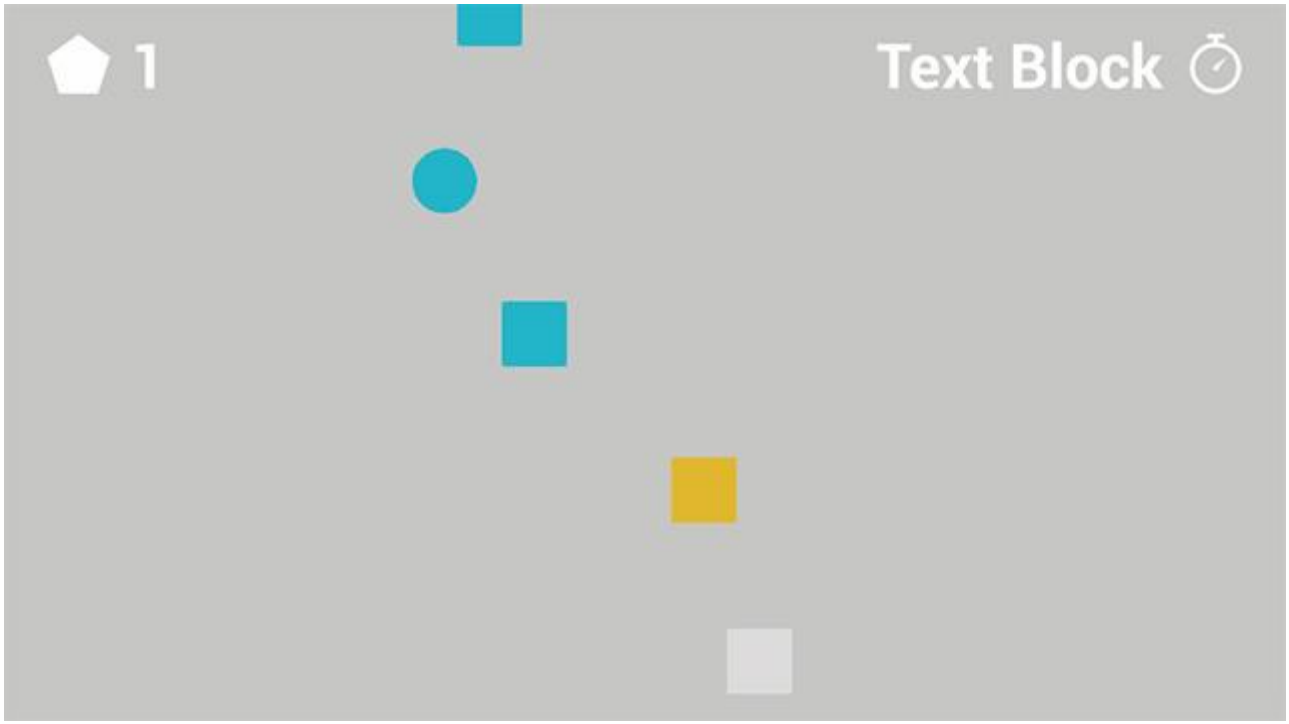
Перетащите переменную *HUDWidget* в граф. *Перетащите* его контакт и *отпустите* на

пустом пространстве. Добавьте нод *UpdateCounterText* и соедините его следующим образом:



Теперь при выполнении *IncrementShapesCollected* она будет увеличивать *ShapesCollected*, а затем вызывать *UpdateCounterText*. Эта функция затем обновит *CounterText* значением *ShapesCollected*.

Нажмите на *Compile* и закройте *BP_GameManager*. Нажмите на *Play* и соберите несколько фигур, чтобы увидеть, как обновляется виджет *CounterText*.

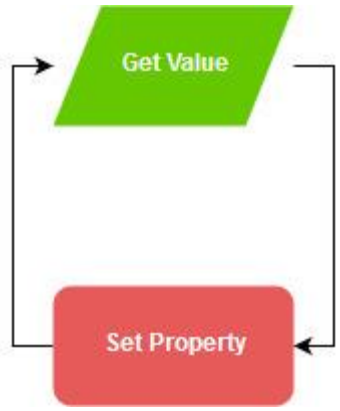


Теперь нам нужно научиться обновлять виджет *TimerText*, только другим способом, который называется *привязкой*.

Bindings

Привязки позволяют автоматически обновлять определённые свойства виджетов. Чтобы иметь возможность привязки, свойство должно обладать *раскрывающимся списком Bind*.

Можно привязывать свойства к функции или к переменной, хранящейся внутри виджета. Привязка постоянно получает значение от функции или переменной и присваивает привязанному свойству это значение.



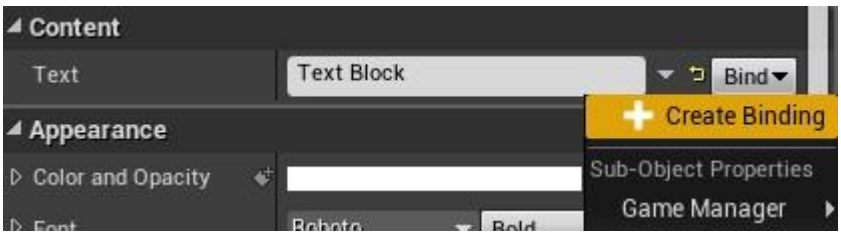
Вы, наверно, задаётесь вопросом, почему же нельзя постоянно использовать привязки. Привязки неэффективны, потому что постоянно обновляются. Это значит, что игра тратит время на обновление свойства, даже если нет никакой новой информации. Сравните это с предыдущим способом, при котором виджет обновлялся только при необходимости.

С учётом этого привязки хорошо подходят для часто изменяющихся элементов, таких как таймеры. Давайте создадим привязку для *TimerText*.

Создание привязки

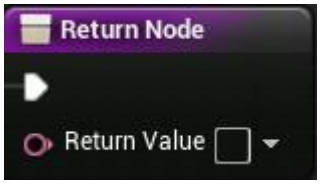
Откройте *WBP_HUD* и переключитесь в режим *Designer*.

Выберите *TimerText*, а затем перейдите в раздел *Content* панели Details. Вы увидите, что свойство *Text* имеет возможность привязки. Нажмите на *раскрывающийся список Bind* и выберите *Create Binding*.



При этом создастся новая функция и выполнится переход к её графу. Переименуйте функцию в *UpdateTimerText*.

Функция будет иметь нод *Return* с контактом *Return Value* типа *Text*. *TimerText* будет отображать любой текст, который вы подключите к этому контакту.



Перетащите *GameManager* на граф и получите из него переменную *TimeRemaining*.

Соедините переменную *TimeRemaining* с *Return Value* нода *Return*. Как и в прошлый раз, Unreal автоматически добавит нод преобразования.

Подведём итог:

- Привязка будет постоянно вызывать функцию *UpdateTimerText*
- Функция получает переменную *TimeRemaining* от *BP_GameManager*
- Нод *ToText (float)* преобразует значение из *TimeRemaining* в тип *Text*.
- Преобразованное значение затем выводится в нод *Return*

Наш HUD наконец готов. Нажмите на *Compile* и закройте *WBP_HUD*. Нажмите на *Play*, чтобы увидеть окончательные результаты.