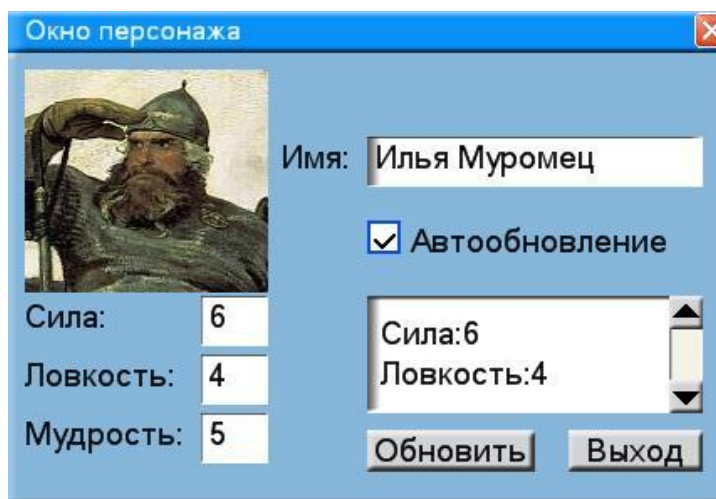


УРОК 1. Создание графического пользовательского интерфейса (GUI).

Примечание: перед компиляцией демонстрационного проекта, вы должны обратиться к папке References, которая находится в дереве файлов проекта, и там указать ссылку на Finis2D.dll из комплекта поставки.

В этом уроке мы научимся создавать графический интерфейс, привычный игроку по обычным приложениям Windows. «Финист 2D» располагает большим набором графических компонент, который позволяет реализовать широкий спектр задач. Если его не достаточно, то можно скачать дополнительные пакеты компонент, где, возможно, найдутся компоненты, нужные именно Вам.

Мы создадим простое приложение, которое, тем не менее, демонстрирует общие принципы формирования пользовательского интерфейса. Задача приложения будет очень проста: требуется реализовать окно ввода характеристик персонажа.



Известно, что персонаж нашей вымышленной игры обладает следующими характеристиками:

1. Имя. Тип данных: строка.
2. Сила. Тип данных: целое число.
3. Ловкость. Тип данных: целое число.
4. Мудрость. Тип данных: целое число.
5. Портрет. Тип данных: картинка.

Для решения этой задачи нам понадобится всего лишь одна форма (окно) с элементами управления. Чтобы обеспечить ввод характеристик персонажа, мы используем такие компоненты:

1. **TextEdit**. Текстовое поле, куда игрок может вводить информацию. Нужно четыре TextEdit (имя, сила, ловкость и мудрость).
2. **Label**. Текстовая метка. Используем четыре компонента как заголовки для пояснения назначения каждого из TextEdit.
3. **ImageBox**. Отображение картинки. Будем использовать этот компонент для отображения портрета персонажа.
4. **ListBox**. Список строк. Мы будем периодически обновлять его, выводя туда сводную информацию о персонаже.
5. **Timer**. Счётчик времени, который срабатывает равномерно через указанный промежуток времени. В тестовом приложении таймер будет задавать частоту обновления списка со сводной информацией о персонаже.
6. **Button**. Нам понадобится две кнопки. Кнопка «Выход» завершает выполнение приложения. Кнопка «Обновить» принудительно обновляет сводную информацию о персонаже, не дожидаясь срабатывания таймера.
7. **Checkbox**. Флажок, включающий/выключающий автообновление.

Создание проекта

Начать работу над приложением нужно с создания нового игрового проекта. Из главного меню «Visual Studio» выполните команду «File -> New -> Project -> Windows Game» (последний пункт меню находится в разделе «XNA Game Studio»).

В открывшемся проекте нужно найти файл проекта, который называется Game1.cs. Переименуйте его в DemoGame.cs. Этот класс унаследован от стандартного Microsoft.XNA.Framework.Game. Нужно унаследовать его от Finist2D.GameProject (добавьте в раздел uses пространство имён Finist2D). Базовый класс обладает массой полезных свойств, о которых можно узнать в соответствующем разделе справки. По ходу проектирования мы будем касаться только некоторых из них.

Поскольку всё необходимое для работы игры реализовано в базовом классе **GameProject**, то из класса, созданного Visual Studio, требуется убрать всё лишнее. В итоге должен остаться конструктор, а также перегруженный метод *LoadContent* с вызовом соответствующего метода базового класса. В более сложных проектах понадобилось бы перекрывать и другие методы базового класса, например, *Draw*, *Update*, *Dispose* и прочие.

Не забудьте вызвать одноимённый метод базового класса в *LoadContent*. Для получения доступа к хранилищу строк, где располагаются все текстовые ресурсы нашей маленькой игры, будем использовать класс **StorageStrings**. Строки находятся в обычном

xml-файле, который мы можем загрузить посредством метода *Load*. Получить строку из хранилища можно по ключу, передав его в метод *GetString*.

Это важно! Обратите внимание, что игра запускается в полноэкранном режиме. Это не очень удобно во время её разработки и отладки. Закомментируйте строчку *Fullscreen = true* в методе *LoadContent*, чтобы предотвратить запуск приложения в полноэкранном режиме.

Создание визуальной формы

В центре нашего приложения будет одна визуальная форма. Другие игровые проекты могут содержать несколько форм, либо не содержать их вовсе.

Хорошей практикой программирования является тематическое разделение классов, входящих в проект. Создадим папку *Forms* в корне проекта. Там мы и разместим классы единственной визуальной формы.

Добавим в папку два класса: **FormGUI.cs** и **FormGUI.Designer.cs**. Подобно обычным формам Windows разнесём реализацию по двум частичным (partial) классам (не забудьте убедиться в том, что два класса находятся в одном пространстве имён). Объявления внутренних компонентов разместим в классе **FormGUI.Designer.cs**, а всю реализацию – в **FormGUI.cs**. Нужно указать, что **FormGUI.cs** унаследован от **Finist2D.GUI.Form**.

Начнём с того, что объявим все нужные компоненты в классе **FormGUI.Designer.cs**.

Затем перейдём в код класса **FormGUI.cs**. Определим конструктор, который принимает на вход экземпляр **GameProject**. В конструкторе создадим все внутренние компоненты. Зададим заголовок формы, текст которого получим при помощи специального вспомогательного класса **StorageStrings**, экземпляр которого мы создали в методе *LoadContent* нашего проекта.

Это важно! Все строковые константы проекта (например, заголовки компонент, сообщения игроку и т.п.) нужно помещать в хранилище, которое находится в поле *StorageStrings* проекта (*GameProject*).

Установим размеры формы через свойства *ClientWidth* и *ClientHeight*. Используем свойство формы *StartPosition*, чтобы установить положение формы на экране при старте приложения. Из возможных значений этого свойства мы выберем *StartPositions.Center*, чтобы поместить форму в центре экрана.

Теперь создадим экземпляр класса **FormGUI** в методе *LoadContent* нашего проекта. Нужно заметить, что проект имеет коллекцию форм, которые автоматически

обрабатываются. Используйте метод проекта *AddForm*, чтобы добавить форму в эту коллекцию.

Загрузка содержимого и настройка компонент

Каждый компонент из набора, предоставляемого «Финист 2D», может быть настроен с точки зрения внешнего вида, чтобы максимально соответствовать теме игры.

Как было указано выше, компоненты формы создаются в конструкторе формы, но их настройка производится в перегруженном методе формы *LoadContent*.

Компоненты, как правило, имеют набор одинаковых свойств, таких как: местоположение на форме (*Location*), *ClientWidth* и *ClientHeight* (ширина и высота компонента соответственно), *Text* (отображаемый текст, если компонент поддерживает его вывод) и многие другие.

Также компоненты имеют некоторые свойства, которые отличают их от других. В нашем примере компонент **ImageBox** имеет метод *LoadPicture*, посредством которого в него загружается картинка. Обратите внимание, что компонент **ImageBox** имеет свойство *DrawSize*, которое определяет область, где будет выведено загруженное изображение. Область должна находиться внутри компонента **ImageBox** и не выходить за его пределы.

*Это важно! Все картинки «Финист 2D» размещает в специальном хранилище. Это экземпляр класса *StorageTexture*, который является полем *StorageTextures* проекта (*GameProject*). При загрузке какой-либо графики нужно взаимодействовать с этим полем.*

Мы загружаем портрет Ильи Муромца в компонент *imgPortrait*. Затем каждому компоненту класса **Label**, то есть текстовой метке, присваиваем её заголовок. Теперь приходит очередь компонент **TextEdit**, полей текстового ввода, куда мы должны вводить параметры персонажа. Разместим их на форме и каждому поставим значение по умолчанию посредством свойства *Text*.

Настала очередь других компонент. Речь идёт о списке строк (**ListBox**), который мы просто помещаем на нужное место, кнопки выхода и обновления (**Button**), флажка автообновления (**Checkbox**) и таймера (**Timer**). У кнопки есть событие *OnClick*, которое срабатывает, когда на кнопку помещается курсор мыши и производится нажатие одной из её кнопок. В обработчик этого события поместим код, который завершает работу приложения.

Мы разместим на форме флажок автообновлений, который включает и выключает таймер. Если таймер выключен, то мы не сможем обновить сводную информацию о персонаже никак иначе, чем нажав на кнопку обновления. Обратите внимание, что свойство флажка *AutoSize* выставлено в *true*. Это означает, что компонент будет изменять

свои размеры автоматически, чтобы вместить отображаемый текст целиком. Поэкспериментируйте с этим свойством, чтобы понять принцип его работы.

Таймер не является визуальным компонентом, в отличие от ранее рассмотренных. Выставим его свойство *Time* равным 5. Это означает, что по прошествии пяти секунд, начиная от момента запуска, свойство *OnTime* этого компонента станет равным true, а также будет вызвано событие *OnTimeRaise*. После этого таймер останется в таком состоянии, потому если мы хотим обеспечить цикличность повторений, то в обработчике события *OnTimeRaise* мы должны выполнить метод таймера *Restart*, который его перезапустит. В этом случае через пять секунд, согласно значению свойства *Time*, таймер сработает опять. Также в обработчик события мы поместим вызов служебного метода, который выведет все строки сводной информации о персонаже в список строк **ListBox**.

*Это важно! Вы не увидите ни один компонент на форме, пока не добавите каждый созданный компонент путём вызова метода *AddControl* в формы. Добавлять компоненты форме следует до её инициализации, то есть сразу после их создания в конструкторе формы.*

И последнее, что нам осталось, это запустить таймер, обслуживающий обновления списка строк с информацией о персонаже. Для этого переключаем его свойство *Enabled* в true, а также выполняем метод *Restart*.

Приложение готово! Давайте запустим его и укажем параметры такого знаменитого персонажа русских былин как Илья Муромец.

В этом уроке мы получили представление об устройстве игрового проекта и визуальных форм, научились создавать и размещать на форме визуальные компоненты, настраивать их свойства и вводить данные с их помощью.