**Муниципальное бюджетное общеобразовательное учреждение гимназия «Пущино» городского округа Пущино Московской области**

**Трудности перевода видеоигр и пути их преодоления**

**Направление: «Информатика и ИКТ»**

**Автор проекта:**

Лебедев Андрей Вадимович,

Учащийся 10А класса МБОУ гимназии «Пущино»

г. Пущино

2023 г.

**Оглавление:**

Введение…………………………………………………………………………………….3

I. Получение доступа к файлам игры……………………………………………………...3

II. Подготовка шрифтов игры……………………………………………………………...3

III. Перевод языковых файлов игры…………………………………………………........5

IV. Перевод имён…………………………………………………………………………...5

V. Поиск текста, не содержащегося в языковых файлах………………………………...7

VI. Корректирование положения текста на экране………………………………………7

VII. Редактирование изображений……………………………………………………….10

VIII. Некоторые проблемы, возникающие при переводе…………………………........11

Заключение………………………………………………………………………………...12

Ссылки……………………………………………………………………………………..13

Приложение………………………………………………………………………………..14

**Введение**

В современном мире всё большую популярность приобретают видеоигры. Сейчас в них играют люди всех возрастов почти во всех уголках мира. Наша страна – не исключение. К сожалению, не все производители игр переводят свои проекты на русский язык. Это в первую очередь касается игр, создающихся небольшими студиями, у которых нет крупных бюджетов на перевод на разные языки. Вместе с тем с ростом уровня знаний английского языка, а также базовых навыков работы с компьютером у людей появилась возможность самостоятельно создавать неофициальные переводы интересных им игр и делиться ими между собой.

В этой работе рассматривается процесс перевода на примере игры **OMORI**, а также трудности, которые могут возникать при переводе игр, от собственно перевода (выбора подходящих слов, перевод имён собственных) до технических трудностей (работы с кодом игры, внедрения перевода в игру).

Цель данной работы – выяснить, какие навыки необходимы переводчику видеоигр, помимо знания английского языка.

**I. Получение доступа к файлам игры**

Все файлы, используемые игрой, как правило, содержатся в закодированном виде. Из-за этого для редактирования файлов их сначала необходимо декодировать. Один из способов декодирования файлов игры **OMORI** – инструмент, созданный пользователем сайта *github.com**Lou1sLх* [1]. Он позволяет сначала декодировать все необходимые файлы, а затем, после внесения изменений, закодировать их обратно.

**II. Подготовка шрифтов игры**

Во многих видеоиграх для написания текста используются общедоступные шрифты. Зачастую они поддерживают широкий набор символов для различных языков. Но иногда авторы видеоигр создают собственные шрифты, которые соответствуют стилистике конкретной игры. Если оригинальный язык игры – английский, то такие шрифты часто включают в себя только буквы английского алфавита и некоторые знаки препинания. Таким образом, для перевода игры необходимо также расширить имеющийся шрифт, добавив в него необходимые символы русского алфавита.

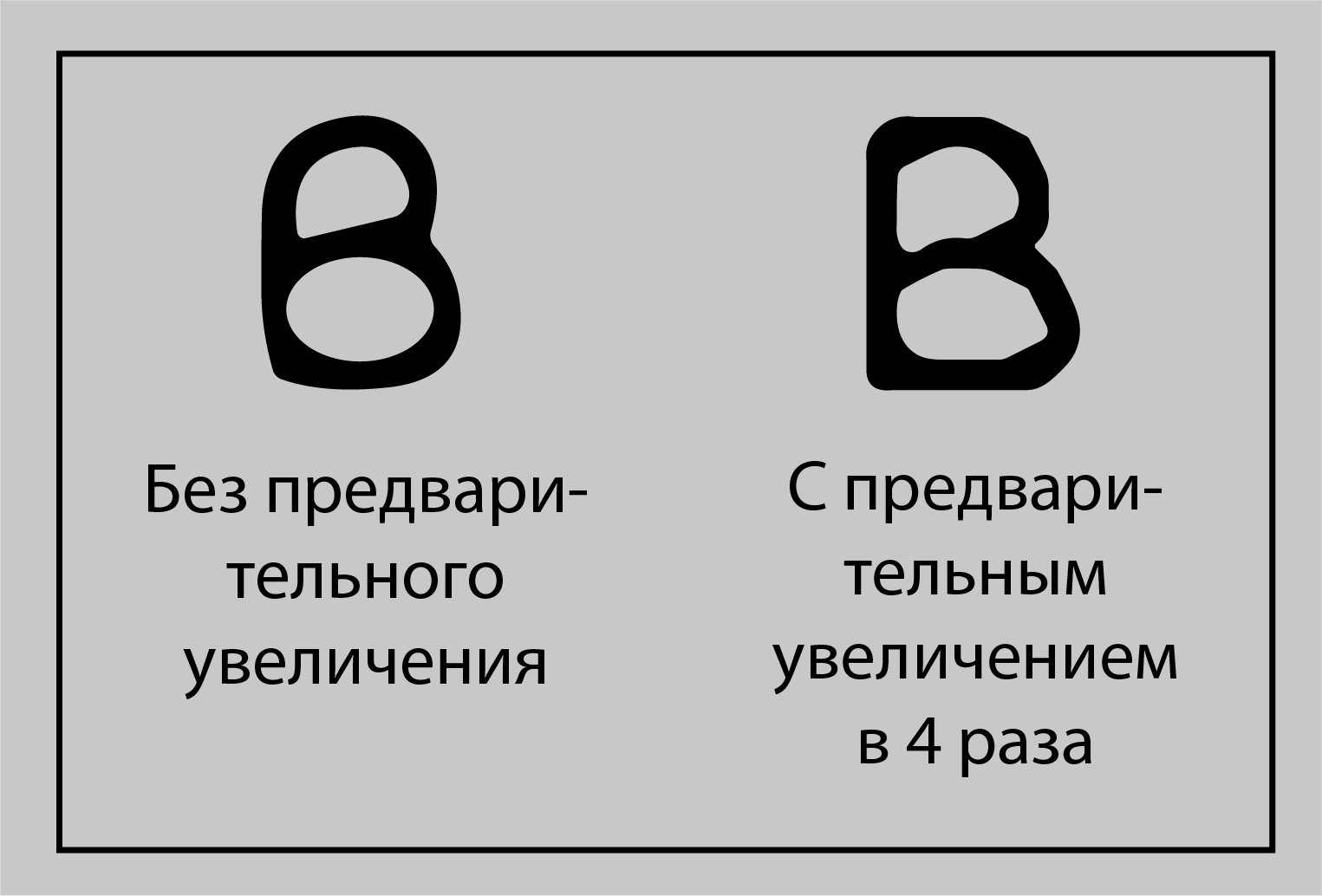
Шрифты игры **OMORI** представляют собой файлы формата *TrueType*. Это один из самых популярных форматов для создания шрифтов. В папке *fonts* корневой папки игры находится 4 шрифта, 2 из которых (*M+1m* и *NotoSans*) являются общедоступными и поддерживают кириллические символы, а 2 других (*OMORI\_GAME* и *OMORI\_GAME2*) созданы разработчиками специально для игры. В качестве примера в этой работе приводится перевод только одного из шрифтов –*OMORI\_GAME2*, который является основным, так как процедура перевода обоих шрифтов одинакова.

Для редактирования шрифтов была использована программа *FontForge* [2]. При детальном рассмотрении отдельных символов можно предположить, что они сначала были нарисованы в растровом виде, а затем преобразованы в векторные объекты (рис 1). Поэтому общая схема создания символов для шрифта такова – нарисовать необходимые символы в любом удобном редакторе изображений, затем любым способом преобразовать их в векторный вид, и добавить в файл шрифта.



*Рис 1. Схема преобразования растровых изображений в векторные*

Для начала все необходимые 66 символов (33 заглавные и 33 строчные буквы), были нарисованы в стилистике исходного шрифта игры. Далее с помощью ресурса *maxlaumeister.com*[3] в 4 раза был увеличен размер изображения каждой буквы, а затем они были конвертированы в необходимый формат (*.svg*) с помощью интернет-ресурса *vectorizer.ai* [4]. Предварительное увеличение размера изображения было необходимо потому, что без него результат оказался неудовлетворительным (рис. 2).



*Рис 2. Обоснование необходимости увеличения пиксельного изображения букв*

**III. Перевод языковых файлов игры**

После создания шрифта и помещения его в соответствующую директорию можно приступать к переводу. Подавляющее большинство текста в игре хранится в папке *…/languages/en* в виде отдельных файлов формата *.yaml* (далее – языковых файлов). Yaml – язык разметки, использующийся для хранения информации о строении и отображении текста. В **OMORI** помимо текста реплик персонажей в языковых файлах хранится информация об эффектах текста (покачивании букв и т.д.), используемом шрифте, имени и портрете персонажа, произносящего данную реплику и др.

Несмотря на то, что в языковых файлах содержится вся необходимая для перевода информация, тексты отдельных реплик были взяты с сайта *goats.dev/omori/* [5], на котором собраны все реплики «в чистом виде» в удобной для копирования и редактирования форме, что значительно ускоряло и облегчало процесс. Подчеркну, что всю информацию для перевода можно брать напрямую из языковых файлов.

Так как длина слов и предложений в русском языке в среднем больше, чем в английском, то при переводе необходимо следить за тем, чтобы весь переведённый текст занимал примерно тот же объём, что и оригинальный. Поэтому изменения в языковые файлы вносились не сразу. Сначала все реплики (оригинальные и соответствующие переведённые) собирались в один текстовый документ. Так легко можно проследить, какая реплика соответствует требуемому размеру, а какую необходимо укоротить.

При переводе игры, как и любого другого произведения, нет необходимости в дословном переводе. Пунктуация, соответствующая правилам русского языка, тоже не так важна. Гораздо важнее передать смысл сказанного, интонации, а также стиль речи, присущий игре и игровым персонажам. При затруднении в переводе были использованы интернет-словари, такие, как *Google Translate*, а также русские словари синонимов, для нахождения наиболее подходящего в определённом контексте слова.

**IV. Перевод имён**

Отдельно стоит отметить перевод на русский язык различных имён. Многие имена персонажей (*Aubrey*, *Shawn*, *Basil*) – реальные имена из разных языков. В этом случае достаточно узнать, как эти имена принято записывать в русском языке. Но это общее правило. На практике же, этим можно пренебрегать в угоду лучшего звучания слова. Так, имя *Basil* в русском обычно записывают как *Бэзил*, но здесь оно было переведено как *Бейзил* (подробнее об этом позже).

Другая группа имён – английские слова либо похожие на них по способу образования конструкции (*Happy*, *Bangs*, *Brows*, *Sweetheart*). Такие имена не следует переводить буквально (*Весёлый*, *Чёлка*, *Брови*, *Дорогуша* хоть и могут хорошо характеризовать персонажей, которым принадлежат, но только запутывают игрока). Правильнее оставлять эти имена непереведёнными, причём записывать их в соответствии с русскими правилами фонетики, а не так, как эти слова звучат в английском. В русском языке принято смягчать многие звуки, что особенно касается устоявшихся иностранных слов. Сравните: *academy* [э к э д э м и] (К и Д твёрдые) и *академия* [а к а д` э м` и й а] (Д мягкий). Поэтому имена *Happy* и *Bangs* были записаны не как *Хэппи* и *Бэнгс*, а *Хеппи* и *Бенгс* соответственно. По этой же причине имя *Basil* было переведено как *Бейзил*, вместо *Бэзил*.

В английском в слове *Brows* буква W не даёт звук [В], а удлиняет стоящий перед ней О (со смещением звучания в сторону [У]). Поэтому более правильная транскрипция – *Броус*. Но в русском 2 гласные подряд могут звучать неестественно и читаться тяжело, поэтому имя было переведено как *Бровс*.

Имя *Sweetheart* состоит из слов *Sweet* и *Heart*. *Heart* (англ. *Сердце*) читается как [х а р т]. Но по незнанию языка многие в России произносят его как [х` ё р т], такое произношение является устоявшимся в русском языке. Поэтому имя *Sweetheart* при переводе было записано как *Свитхёрт*.

Имена, вызывающие чуть больше проблем для перевода – различные выражения, характеризующие персонажа и его деятельность. Характерным примером таких имён являются *Capt. Spaceboy* (букв. *Капитан Космомальчик)* и *Space Boyfriend* (букв. *Космический парень*). Придумать перевод, который не звучал бы неестественно и искусственно, не вызывал бы у игрока улыбку своей нелепостью, достаточно сложно. Тут всё зависит от творческих способностей и фантазии переводчика. После многочисленных вариантов было решено остановиться на именах *Звёздный паренёк* для *Capt. Spaceboy* и *Космо-Возлюбленный* для *Space Boyfriend*. Можно спорить о корректности такого перевода и о соответствии его вышеперечисленным критериям. В любом случае, сложно оценить его качество без участия независимого игрока-тестировщика, который может выразить своё мнение во время прохождения игры.

Приведённый пример – сложное для перевода имя. Пример гораздо более простого имени – *Capt. Pinkbeard*. Очевидным переводом является *Капитан Розовая Борода*.

Самые сложные для перевода имена – имена, включающие в себя игру слов в том или ином виде. Это, как правило, имена различных врагов. Трудность перевода названий этой категории сильно различается от случая к случаю – хорошее имя может прийти в голову сразу, а иногда приходиться думать над ним несколько дней.

**V. Поиск текста, не содержащегося в языковых файлах**

Большое количество текста содержится в других, помимо языковых, файлах. Это, как правило, элементы интерфейса, описания предметов, способностей персонажей, атак врагов. Для перевода всей этой информации необходимо сначала найти место её расположения. В игре содержится большое количество файлов. Причём некоторые из них, подобно файлам формата *.yaml*, выступают в роли носителей информации и почти полностью состоят из простого текста. Другие же файлы написаны на языке программирования, трудном для понимания, занимают несколько тысяч строк и при этом содержат в себе незначительное количество текста, который всё же необходимо найти, выделить и перевести. Поэтому поиск нужной информации среди всех файлов может стать нетривиальной задачей, особенно для тех, кто плохо разбирается в основных структурах языков программирования и для кого весь код в файлах представляется просто массой ничего не значащих символов.

**VI. Корректировка положения текста на экране**

При переводе можно столкнуться с рядом неприятных проблем: текст выглядит несимметрично, не помещается в отведённое ему место, расстояния между отдельными словами различны. Это, как правило, касается элементов интерфейса. Они располагались на экране исходя из длины слов в английском языке, и поэтому их расположение может не подходить для русских слов. Большинство программных файлов, в которых создаются сцены игры и размещаются все объекты, в том числе и текст, находятся в папке *../js/plugins* (место расположения подобных файлов, очевидно, различается для разных игр). К счастью, по названию файла зачастую можно догадаться о его функции. Например, кнопка «Новая игра» на начальном экране рисуется в файле *Omori Title Screen.js*.

Содержимое файлов может занимать очень большой объём, и просматривание всего файла в поисках нужной операции, особенно человеку, не знакомому с языком программирования, может показаться очень сложной задачей. Для этого программисты, пишущие объёмные проекты, оставляют в коде большое количество комментариев, то есть строк, игнорируемых программой во время исполнения, но подсказывающих людям, для чего предназначена та или иная часть кода (рис. 3). В **OMORI** отрисовка почти каждого изображения на экране сопровождается комментарием. Поэтому для нахождения определённого объекта достаточно знаний английского языка (которые у переводчика наверняка есть) и базовое понимание лексики, связанной с программированием.



*Рис 3. Часть файла игры. Зелёные строчки, начинающиеся с* ***//*** *- комментарии*

Пример работы с центрированием текста:

Ниже представлена часть кода, отвечающая за отрисовку трёх кнопок на главном экране – «Новая Игра», «Продолжить» и «Настройки»:

*381* **//=======================================================================**

*382* **// \* Create Title Commands**

*383* **//=======================================================================**

*384* **Scene\_OmoriTitleScreen.prototype.createTitleCommands = function() {**

*385* **// Initialize Title Comands**

*386* **this.\_titleCommands = [];**

*387*  **// Text Array**

*388* **var textList = LanguageManager.getMessageData("XX\_BLUE.Omori\_Title\_Screen").commands**

*389* **// Get Center X Position**

*390* **var centerX = Math.floor((Graphics.width - (156 \* textList.length)) / 1.8)**

*391* **// Go Through Text Array**

*392* **for (var i = 0; i < textList.length; i++) {**

*393* **// Get Text**

*394* **var text = textList[i];**

*395* **// Create Window**

*396* **var win = new Window\_OmoTitleScreenBox(text);**

*397* **// Set Wnidow Position**

*398* **win.x = centerX + (i \* (130 + 39));**

*399*  **win.y = Graphics.height //(Graphics.height - win.height) - 22**

*400* **// Select Window**

*401* **if (i === this.\_commandIndex) { win.select(0)}**

*402* **// Add window to title Commands**

*403* **this.\_titleCommands[i] = win;**

*404*  **this.addChild(win)**

*405* **};**

*406* **// Set Continue text**

*407* **this.\_titleCommands[1].setText(textList[1], this.\_canContinue);**

*408*

*409* **};**

Первые 3 строчки-комментария – «заглавие» блока команд. Они чётко дают понять, за что он отвечает – создаёт команды заглавия (англ. *Create Title Commands*), то есть 3 вышеназванные кнопки. Пусть переводчику кажется, что все кнопки слишком смещены вправо по горизонтали (или по оси X), поэтому он хочет сместить центр, относительно которого рисуются кнопки, влево по оси X. Можно догадаться, что расчет положения центра производится в строке *390*, так как в строке *389* содержится комментарий – *Get Center X Position* (англ. *Получить Позицию Центра по X*). Подставив различные значения, человек, работающий с кодом, может решить, что в скобки необходимо дописать значение *-45*, что означает смещение центра на 45 пикселей влево.

Также переводчик хочет расположить 3 кнопки на большем расстоянии друг от друга. Из строчек *387-388* он может заметить, что информация о кнопках хранится в массиве – типе данных в программировании, в котором может храниться набор однотипных элементов, в нашем случае, текста кнопок. Дальше в цикле *for* (строчки *392-405*) программа проходит по этому массиву и создаёт объект с заданным текстом для каждой кнопки. Причём расположение каждой кнопки задаётся в строчках *397-399* (англ. *Set Window Position* – *Установить Позицию Окна*). Это расположение зависит от переменной *i*, увеличивающейся при каждом проходе цикла, и некоторого множителя *(130 + 39).* Увеличение этого множителя увеличивает расстояние между кнопками. Подбором разных вариантов можно прийти к выводу, что добавление к множителю числа *+30* даёт требуемый результат.

Пусть переводчик считает также нужным отдельно сдвинуть центральную кнопку «Продолжить» левее. Из строчки *407* можно видеть, что возможно отдельно изменить параметр любого элемента массива (*this.\_titleCommands[1]* – обращение ко 2 элементу массива, то есть к объекту кнопки «Продолжить», так как нумерация элементов в большинстве языков программирования начинается с 0). Из предыдущих строчек можно также понять, что атрибут *.x* отвечает за положение на экране по оси X (Видно, что программа обращается к атрибуту *.x* в строчке *398*). После цикла *for* можно дописать дополнительную строчку *this.\_titleCommands[1].x = this.\_titleCommands[1].x - 5*. В ней от текущего значения атрибута *.x* 2-го элемента массива кнопок меню отнимается 5, сдвигая кнопку «Продолжить» на 5 пикселей влево.

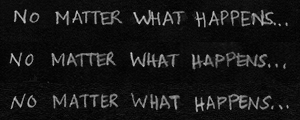
В некоторых случаях, когда нельзя так просто изменить положение требуемого окна с текстом, или в случае недостаточных навыков программирования, можно добавить в конец или в начало текста несколько пробелов. Это поможет сместить текстовое окно, но также может вызвать некорректное отображение некоторых элементов интерфейса (в частности, курсора, показывающего выбранное действие), которые ориентируются на координаты объекта, а не на положение значащих символов текста. Но, как правило, это малозаметно при игре.

Если никак не получается написать текст короче, а он всё равно не помещается в отведённое ему место, можно уменьшить размер шрифта, отображаемого в этом окне. Но, чтобы найти место хранения этого размера, необходимо также внимательно изучать код игры.

**VII. Редактирование изображений**

Большое количество текста в игре также содержится в виде изображений. Это в первую очередь многие элементы интерфейса, а также некоторые сюжетно важные предметы. Для изменения этого текста подойдёт любой редактор изображений, например, *Adobe Photoshop*. Так как в игре разрешение экрана (а значит и всех изображений) небольшое, то и размер каждой отдельной буквы текста, как правило, также маленький, и все буквы удобно перерисовывать попиксельно. Мало что можно сказать про этот процесс: нет общего правила, так как каждый отдельный случай индивидуален.

Отдельный интерес представляют *катсцены* (от англ. *сutscene* – эпизод в видеоиграх, в которых игрок наблюдает за происходящим на экране и не может влиять на ход событий, проще говоря, *внутриигровое видео*). В **OMORI** они представлены в двух видах: как заранее созданное видео, и как последовательность отдельных изображений, которые воспроизводятся во время игры. В этих катсценах, как правило, показываются несколько изображений, сменяющих друг друга каждые несколько секунд. Чтобы избежать статичности, каждое изображение нарисовано несколько раз с незначительными изменениями: кадры сменяют друг друга несколько раз в секунду, создавая ощущение движения (рис 4).



*Рис 4. На 3 разных кадрах строчки незначительно отличаются*

При редактировании таких изображений удобно сначала создать текстовый объект (возможно почти во всех редакторах изображений), поместить его на нужных местах в кадрах, а затем уже поверх него рисовать вариации символов, используя созданный текст в качестве основы. Так все буквы получаются ровнее, чем если бы рисовались независимо друг от друга.

Катсцены, хранящиеся в формате видео, можно редактировать покадрово.

**VIII. Некоторые проблемы, возникающие при переводе**

Одной из проблем, которая совершенно неожиданно обнаружилась во время работы над файлами игры, была их плохая систематизация. Несколько тому примеров:

Изображение одного из главных геймплейных элементов интерфейса – клавиш *Атака/Защита/Закуски/Игрушки* содержится в том или ином виде в 4 различных файлах в разных директориях, при этом самой игрой используется только один. Выяснить, какой именно, можно только попробовав поочередно изменить все 4 файла.

Кадры начальной катсцены игры содержатся в 5 файлах. При этом все 5 файлов не носят однотипные названия (например, *Intro\_Cutscene\_1.png*, *Intro\_Cutscene\_2.png*, *Intro\_Cutscene\_3.png* и т.д.), по которым легко было бы определить, принадлежат ли они к одному игровому событию. Поэтому необходимо было несколько раз просмотреть все изображения в игре, чтобы определить все необходимые изображения.

Выраженность этой проблемы может зависеть и от сложности игры, и от расторопности разработчика, её создающую. Про неё не стоит забывать, начиная работать над переводом игры.

Другая значительная проблема – склонность к недооценке времени, требуемого для перевода определённого объёма текста. Причина её, скорее всего, в следующем: человек воспринимает информацию (в том числе зрительно) гораздо быстрее, чем может её обратно воспроизвести. Поэтому тот объём текста, прочтение которого при прохождении игры занимает несколько минут, может потребовать несколько часов (включая время на поиск расположения этого текста, перевод и иногда необходимые коррекции) для перевода и внедрения в игру. Это обстоятельство также следует иметь в виду.

**Заключение**

По результатам этой работы можно прийти к нескольким выводам: во-первых, перевод игры требует от переводчика обширных знаний не только в области иностранного языка, но также азов программирования, рисования, умения самостоятельно искать нужную информацию. Во-вторых, перевод игры – это очень времязатратно. От человека требуется большое терпение и усидчивость.

Но при этом нужно отметить, что создание перевода может дать переводчику пользу, например, повысить уровень владения английским языком. Не стоит забывать и про повышение навыков краткого выражения мыслей на родном языке. Вообще же перевод игры, особенно той, которая интересна переводчику – очень занимательный процесс. Узнать что-то новое об уже пройденной игре, пока разбираешь её код, лучше понять программы и то, как их создают профессионалы – всё это очень интересный опыт, который автор работы рекомендует ощутить каждому, интересующемуся видеоиграми и тем, как они работают изнутри.

**Ссылки:**

1. Инструмент для декодирования файлов игры – <https://github.com/Lou1sL/omori_decrypt>

2. Редактор шрифтов *FontForge* – <https://fontforge.org/en-US/>

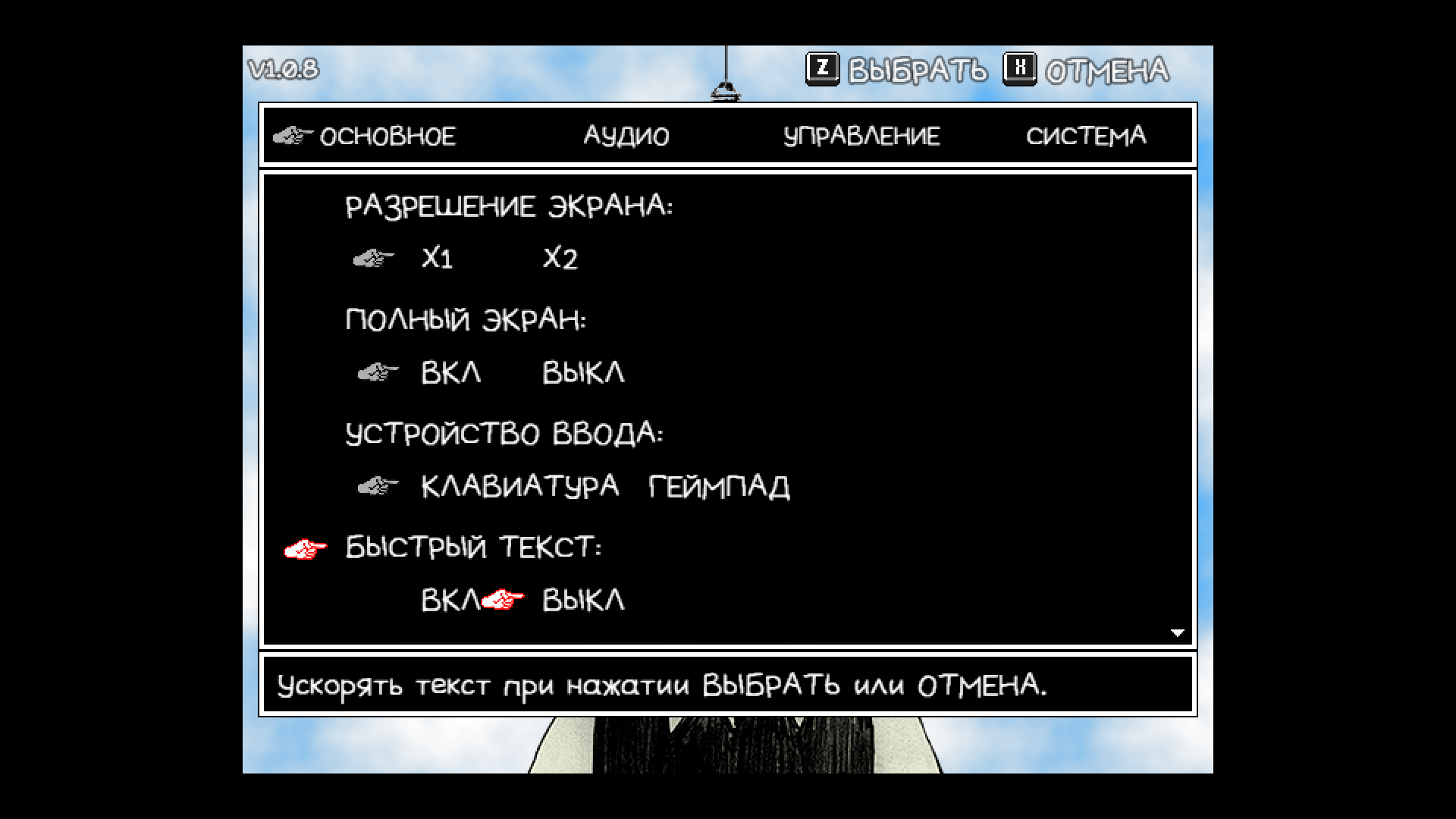
3. Апскейлер (увеличитель разрешения изображений) – <https://www.maxlaumeister.com/pixel-art-upscaler/>

4. Векторайзер (преобразователь изображений в векторный формат) – <https://vectorizer.ai/>

5. Все реплики персонажей и диалоги из игры – <https://goats.dev/omori/>

**Приложение: несколько кадров из переведённой игры**

****

****

*Главное меню игры*

**

**

**

**