University of Glasgow | School of Computing Science

Volcanic: an Open Source SimAntics IDE

Rhys Simpson

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 7, 2016

Abstract

Custom objects for The Sims have been a main part of its success for a long time, and have potential future interest thanks to efforts aiming to revive its spinoff, The Sims Online. However, with the best tools available being painful to use, closed source or unreleased, its famous custom content legacy and impressively complex "SimAntics" scripting environment risk fading into history.

Introducing Volcanic, a new open-source alternative for creating objects in The Sims Online, which aims to unify the functions of the existing tools into one fully Integrated Development Environment. The tool utilises an open source re-implementation of the game engine, "FreeSO", to provide extensive live debugging features not possible before. Simply through its existence, the tool has already inspired long-time The Sims fans to revisit content creation, despite just getting started.

Acknowledgements

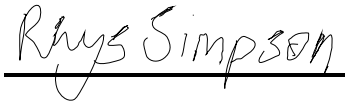
I would like to thank my project supervisor, Simon Gay, for his continued support and advice throughout the 6 month period. A lot of helpful feedback was provided by the FreeSO community, even before the evaluation was being run, which proved invaluable in helping the project reach its current state.

I would also like to thank Darren Lee, Andrew D'Addesio and Mats Vederhus for laying the foundations for the re-implementation of The Sims Online, without which this project would not be possible. The resources and reverse engineering work done by SimTech and other The Sims communities for The Sims 1 a long time ago have also been greatly responsible for the advances in the re-implemented engine.

Finally, I'd like to thank all 9 dedicated participants of the evaluation. Many of them went over and above with their feedback; it is these users with the same passion for modding The Sims that I'd like to show my deepest gratitudes towards.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Rhys Simpson Signature: 

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 The Sims, The Sims Online and FreeSO

(a) The Sims (from The Sims Wikia)

(b) The Sims Online (from ModDB)

(c) FreeSO

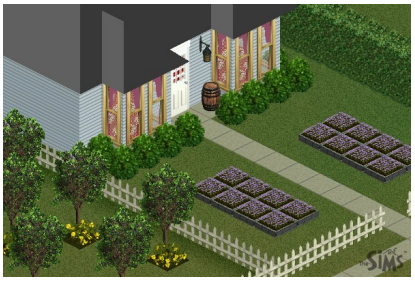
"The Sims" is a life simulation developed by Maxis, originally released in the US on February 4th, 2000 [1]. The gameplay features three distinct, but linked modes: Buy Mode, Build Mode and Live Mode. In "Build Mode", players can construct buildings out of walls and floors, then add doors, windows, fences etc. In "Buy Mode", players can populate these buildings with household "Objects", each with its own unique appearance, placement rules and functionality in "Live Mode". In "Live Mode", players are put in control of "Sims", simulated people who live in the building the player has created, and need to manage their Motives, improve their Skills, and propel them up to the top of the Career ladder. This is achieved by interacting with the household Objects placed in "Buy Mode", each having their own unique purpose. This could be anything from replenishing a need, to increasing a skill, to causing the death of a particular sim. Other than micromanagement of motives and the arbitrary goal of furthering your sims' careers and earning money, The Sims is a "sandbox" game which has no end goal and allows the player to do whatever they feel like. Arguably, the gameplay is *entirely defined by the objects that the player can use*, and the sims' interactions with them.

The Sims was met with much praise, gaining a current Metacritic score of 92/100 [2], and more importantly overtaking Myst in 2002 to become the best selling PC game at the time [3]. Due to this success, Maxis saw promise in an "Massive Multiplayer Online" (or MMO) variation of The Sims formula. They released a spin-off game called **"The Sims Online"** in 2002, where instead of controlling a family of Sims, players controlled a single sim in an online world populated by only other player-owned sims. In The Sims Online, the gameplay is much more defined by the interactions between players than the Motive, Skill and Career focussed areas. These areas did still exist, though they were modified to encourage more user involvement, and made it possible to collaborate with other players though group oriented object interactions to achieve their personal goals.

Due to a lack of success, The Sims Online lay dormant for several years, but was later renamed to **"EA-Land"** in 2007 in an attempted reboot which restructured the economy, introduced a "Free to Play" model and begun accepting Custom Content into the game. This did not take off, and The Sims Online closed down for good on August 1st, 2008.[4] Since the closure, the dedicated community of The Sims Online has attempted to revive the game multiple times, with the current effort being **"FreeSO"**, an Open Source re-implementation of The Sims Online using C# and Monogame [5]. This engine reads the same resources used by the original game, including all objects, sounds, scripts and graphics, and presents them in a similar fashion. These resources must be provided by the user, to avoid FreeSO itself infringing on any copyrights. FreeSO is quickly nearing completion - supporting most objects in the game and rudimentary Server-Client online play. It is currently the only active open source re-implementation of the engine used in The Sims and The Sims Online, making it *the only target for a tool which hopes to interface with a live instance of the game environment.*

1.1.2 Custom Content in The Sims

(a) simlogical "Winter" season



(b) simlogical "Summer" season

(c) Scan of the PC Gamer Article on the "Slice City" set of custom objects.[6]

The Sims was designed in such a way that made it easy for them to add new household objects to the game - the objects were entirely self contained (graphics, metadata), used bundled scripts to provide functionality, and interfaced with each other within a controlled environment. This allowed Maxis to easily create a number of "expansion packs", which expanded the gameplay by simply adding new objects with some minor engine changes to add new over-world areas, or features for objects to utilise. Early in the game's life-cycle, dedicated players caught onto this practice, and discovered that not only object graphics, but their scripts were also entirely self contained, and run in-game using a bytecode language referred to as SimAntics. Aside from some tools for recolouring objects, walls and floors, Maxis themselves did not release any of the tools used internally to develop the official objects, so the community produced their own suite of tools by reverse engineering the formats used by the game.

These tools were ultimately quite rough, as will be explained later. Despite this, the Sims modding community became an important part of The Sims's success, as they were able to make substantial, unique and beneficial changes to the gameplay. With content creators eventually creating full, uniquely designed object sets, players were spoiled for choice for how to design the rooms in their houses. One site, SimLogical, went as far as to mod a set of game objects to simulate visual seasons in the game, and created full gameplay controllers which simulated various institutions such as prisons and schools [7].

Entire mini-games have even been created in The Sims, such as SimSlice's "Slice City", which introduces "SimCity"-like gameplay within sim households, and was even covered by news outlets such as PC Gamer and the BBC [6] [8]. This area only just started being explored by the time The Sims 2 released; there is still a lot more possible with SimAntics and with The Sims Online, which never saw any object modifications in its lifetime. The question is, what could players create with full control of the multi-player focused environment in The Sims Online?

1.2 Aim

The purpose of this project is to develop a fully integrated development environment for SimAntics, that allows a user to fully create an object for The Sims Online from start to finish. The final product should allow a user to view and substantially modify existing object resources, see and test changes to objects immediately in the game, and utilize the game environment to "step through" the object scripts primitive by primitive to trace execution. Ideally, the user would not have to use any external programs to modify an object - it should be possible to perform the full range of possible changes without closing the program, and while staying in the live debug game environment.

The goal is to have a tool which is ultimately both more usable and more powerful than the current alternatives for The Sims 1. As will be explained, these tools are either incomplete, fractured, or inaccessible - and they do not support the modified environment that The Sims Online runs on. The tool should support editing and creating most resource formats used by objects to permit substantial changes.

1.3 Motivations

The main motivation is to allow users to experiment with and make game objects using SimAntics, either out of interest or a desire to improve the game experience for both themselves and others. These game objects could be developed for use with the re-implementation project FreeSO, to logically expand upon the gameplay of The Sims Online past what Maxis initially intended in a community driven fashion.

It should also be able to patch existing objects, in a fashion that clones them for the purpose of recolouring, or modifies them in a way that does not affect the original files. These patches could simply include extra skins for existing object, such as chairs and tables, but they could also include changes and additions to their functionality to fix bugs or add new features. By storing changes to a game file as a patch instead of a duplicate copy that has been modified, changes to files used in the original game could be distributed without re-distributing any of the copyrighted content included with the base game. (Video game ROM hacks generally utilise this kind of patching, with the .IPS format.)[9] This means that gameplay changing patches to objects could technically be distributed to all players connected to a central server, without legal implications.

With the ability to modify game resources as well as the engine itself, the tool would open up the possibility of utilizing the SimAntics engine for other purposes, such as an entirely standalone game. The benefits of doing this would be the ability to utilize the modularity of The Sims objects, specifically in games where there need to be many unique items to interact with, such as point and click adventure games.

A tool like this would definitely be useful for historical purposes alone - many people such as myself grew up with the game, and might like to know how it worked behind the scenes. Those interested in The Sims instead of The Sims Online would still be able to achieve this goal, since both games share many of the same objects. Though the existing tools do fulfil this purpose to an extent, they do not directly link with the game to allow players to interact with the scripts at runtime.

Chapter 2

Primer for the SimAntics Environment

2.1 The Sims and "Objects"

As previously mentioned, the gameplay of The Sims is centred around distinct household **"Objects"** - which are typically placed using "Buy Mode" - and the simulated characters' interactions with them. Objects are primarily there for sims to interact with, but they can also interact with each other and change by themselves, such as plants wilting over time.

Objects typically have a sets of graphics, interactions, functions and other resources associated with them. These include string resources for dialog popup text, names for sim animations that the object uses, routing destination position information among other things. The most important resource in an object is its **"Object Definition"** which defines various functional and physical properties of the object, as well as defining "entry point" scripts that the object should run at well defined points in time (eg. on creation: init, constantly: main). All of these elements affect how the objects are presented in the world, and drive the logic behind how sims and other objects interact with them.

From a technical perspective, Objects in The Sims Online are not necessarily furniture - the game uses many invisible "controller" objects to control complex game elements. These stem from things like career/skill progression, to spawning NPCs like a maid or gardener on a schedule, to even entirely driving multi-player "Job Lot" mini-games. In fact, in The Sims, *the sims themselves are technically objects of a special kind* - their idle animations, low motive reactions such as death and even "go here" requests are driven by the same scripting language as "regular" objects. NPCs in the game, such as a maid character who you can hire to clean up after your sims for a periodic fee, are simply "person objects" with a different script driving their idle loop.

Objects need not only take up just one tile either - objects can be **"multi-tile"**, meaning that they are actually made up of many single-tile "parts" tied together by a "master definition". This defines its appearance in the catalog and high level attributes, while the single-tiles define their own functional and physical properties.

2.2 The IFF Content System

The modularity of objects in The Sims comes primarily from the use of their container format, **"Interchange File Format"**, or **"IFF"**. This format is used to package multiple **"chunk"** resources of vastly different types in one file, that manages all of its internal cross references purely by using 16-bit integer IDs given to each chunk. Each chunk in an IFF file has this ID, a name, and a specified data type. The data contained within any chunk can be

of any format, so long as the IFF reader knows how to read it correctly (if not, it can be skipped over). It's worth noting that two chunks of different data types can share the same ID, as they are identified by a combination of their ID and type eg. "BHAV #4096" and "TTAB #4096".

The Sims uses this format to group the resources used by objects so that they are both self contained, and locally reference-able from each other. Object IFF Files contain **"Object Definition"** or **"OBJD"** chunks, which define unique objects contained within that IFF file, and point to the main resources that they use, such as their interaction and animation tables. Object IFFs can actually contain multiple objects, simply by including a number of these OBJD resources. Due to this behaviour, it is possible for multiple objects to share resources within the same file - for example a number of chairs with completely different appearances could share the same set of interactions, scripts and strings. The purposes of all chunks used in objects will be later explained in the requirements section.

When resources reference each other from within the same file (eg. BHAV #4096 shows a dialog in STR# #130), this is called accessing the **"Private"** scope. However, Object IFFs in The Sims introduce two additional scopes which select resources can access: **"Global"** and **"Semi-Global"**. The "Global" resources are the same for every object in the game, and contain general purpose strings or scripts that are used by most objects. "Semi-global" resources are different depending on the Semi-global IFF that the Object defines with a **"GLOB"** type chunk. These resources are typically meant to be shared between objects of a set "type", such as sharing functionality for "door" type objects across multiple Object IFFs (doorglobals.iff). This provides a rather rudimentary way to create "classes" of objects which individual IFF files can "subclass" and extend upon. This functionality is used extensively throughout The Sims Online for sets of related objects, such as doors, chairs, tables, lights, and even job objects.

2.3 The SimAntics Virtual Machine

Since objects are designed to be modular, they must be fully managed in a way that lets them safely interact with each other, and contain their own functionality. To achieve this, the Sims uses a custom Virtual Machine and scripting language dubbed **"SimAntics"**.

Each ingame household runs a **SimAntics VM** as its main driving force - which can contain and simulate an arbitrary number of Object Instances. Each one of these instances has its own internal state - specifically "Object Data", and arbitrary per-object defined "Attributes" which can contain any state the object needs to remember for its unique function. Instances also have their own "SimAntics Thread", which drives the currently active script on that object.

Person Objects contain some extra data, one collection of which is called "Person Data" and contains information like Skill levels, and the other is called Motives. Person objects can also run a special kind of stack frame on their thread which routes the sim through the property, generally invoked by a Goto Relative or Goto Routing Slot primitive. They also contain some rudimentary state to define their default "outfit", and state to manage the animation the sim is currently playing.

The SimAntics VM also manages data relating to the lot architecture, such as the walls and floors in a lot. This information can be accessed by the VM in a variety of forms - the most common being placement validation, where on movement the object's destination position is checked for any violations to the placement rules set by its SimAntics trees (eg. cannot intersect person, wall not allowed on right). This information is also used in determining which "room" a certain object in, which is used for lighting, and can be queried at real time to find out if walls on specific directions are adjacent to the object. Finally, the information is used for generating per-room collections of rectangular physical obstacles for use by sim routing and object placement checks, populated by objects, sims, walls and room boundaries.

.

2.4 BHAV Scripts and Interactions

Object functionality in The Sims is entirely driven by **BHAV** script chunks within their IFF file. These contain **SimAntics bytecode**, can be called by each other, interactions and as "entry points", and define the set of Parameters and Local Variables that they expect/use. The most common representation of SimAntics for editing is a node graph, generally in the shape of a tree. Each node is a "Primitive"; an atomic operation which performs some function and returns true or false. These functions can range from simply setting one variable to another ("Expression"), to handing over control to a "Routing Frame" to route the Caller sim to a destination location.

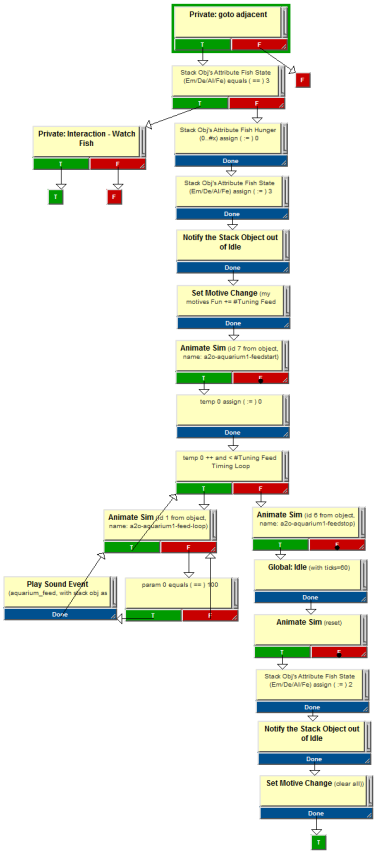


Figure 2.1: An example BHAV Tree in Edith, which animates a sim feeding fish in an aquarium.

SimAntics threads use a stack based approach to function calls - when a BHAV is run on a thread it is given a **"stack frame"** which has its own internal state. Each stack frame on a SimAntics Thread contains its own parameters, local variables, a reference to the active BHAV routine and its current instruction position. This state also includes references to various objects: The "Caller" or "Me" object (generally the object that owns the thread), the "Callee" object (the one that owns the executing BHAV) and the "Stack Object", an object variable which primitives access to modify the state of other objects. This "Stack Object" can be changed simply by setting its target Object ID with the expression primitive, or using specific primitives such as "Set To Next" to find specific objects in the entire household. Through these object references, objects can easily access and modify each other's state in meaningful ways.

When new BHAV subroutines are pushed onto the stack, the Stack Object is inherited from the parent stack frame, and the parameters of the new frame are either copied from the Temp registers of that thread or (when available) the parameters specified by the subroutine caller. Stack frames also return true or false, which can define which primitive the caller stack frame goes to next.

Technically, primitives actually have more return values than True and False - they can also return "Continue on Next Tick", and "Error". "Continue on Next Tick" yields the thread's execution until the next SimAntics tick, and is generally called by primitives such as the "Sleep" and "Animate Sim" primitives, and also internally by sim routing frames. When the "Continue" return type is used, the instruction pointer does not advance from the active primitive, and will instead run the primitive again when execution of the Thread resumes. Primitives returning "Error" is SimAntics' way of throwing exceptions - though in FreeSO this functionality is not implemented and instead replaced with direct C# exceptions.

Interactions in The Sims are not directly defined by the BHAV resources an object contains. Instead, object interactions are defined by a "Tree Table", or "TTAB", which contains an arbitrary number of named interactions. These can be queued on a given object thread (generally a sim), either by the SimAntics primitive "Push Interaction" or by the player onto their sim via a UI construct known as a **"Pie Menu"**. The first item in this queue can be run on its object's stack when their main routine calls "idle for input with allow push", at which point it is also removed from the queue.

(a) "Pie Menu" ingame. (b) Categories, defined by names split by "/". (c) Flags of the "Set Tree State" interaction.

Figure 2.2: Interactions in The Sims Online.

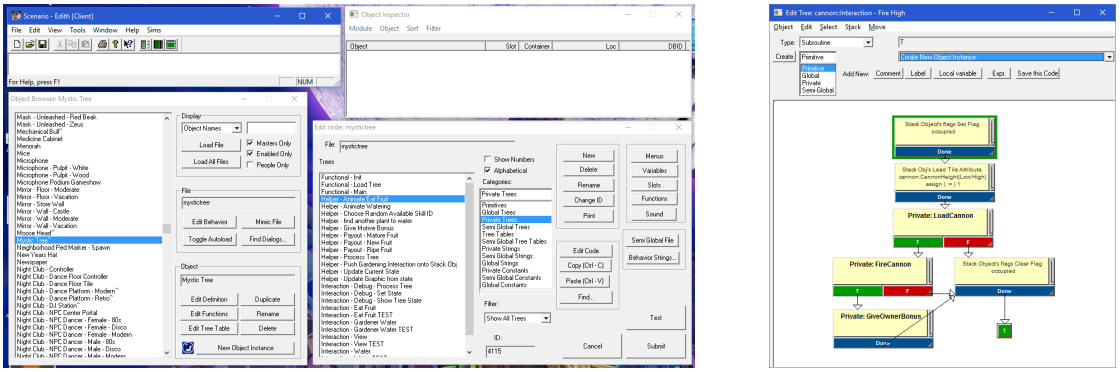
These interactions reference "action" and "check" BHAV trees. The "check" tree is run immediately when the player opens the "Pie Menu", and it determines whether or not that interaction should be displayed in it (by its return value - true or false - and the value of the "hide interaction" object data field). The action tree runs when the interaction is pushed onto the object's stack, and has its Caller to the owner thread and Callee + Stack Object to the target object. Interactions also define a number of flags associated with them, which limit the sims that can queue them, and can also change the behaviour on the interaction queue (eg. run immediately, leapfrog to front of queue).

Ultimately, the scripting language in The Sims is powerful enough to drive gameplay entirely by itself - which is actually what the game does! Much of the gameplay is built around "controller" objects which run persistent scripts in the background, and the sims themselves run BHAV scripts to manage their idle animations, motive failure handling and running queued interactions. Without SimAntics, households would be entirely static, and not even sims would move. There are many more complicated features of the SimAntics VM, such as Create Object Instance "Interaction Callbacks", and looking up and running subroutines by name, on any arbitrary object (selected by Stack Object), on their stack instead of the caller's! These are not necessary to understand the scope, purpose and function of the IDE, so they need not be covered.

Chapter 3

Context

3.1 Related Work



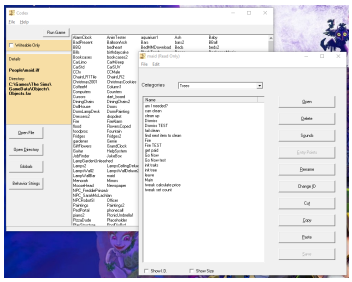
(a) Main Window, Object Browser/Inspector, IFF Resource Viewer (b) A simple BHAV viewed in Edith.

Figure 3.1: Screenshots of the internal object editing tool, Edith.

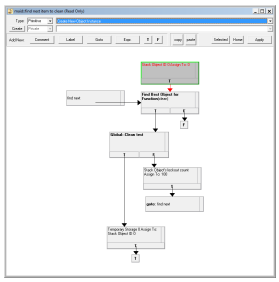
Edith is the official editor for objects in The Sims. It is an internal tool, developed alongside the game itself, which allows users to edit various resources at runtime and note changes to them instantly ingame. All game resources are editable in Edith, except from the graphical ones: Palettes, Sprites, and Drawgroups (PALT, SPR2 and DGRP). It also allows the user to set breakpoints in BHAV trees, which trigger when an ingame object runs into them, opening a "Tracer" window. This allows the user to step through the script execution of an object primitive by primitive, with a full editable view of the object state available. The editor does this by tying into a live instance of the game, and even allows you to view and edit information about the target SimAntics's VM's global state. The tool's design is rather rough, as it was never officially released to the public.

However, since Edith was only used by internal Maxis object developers and never released to the public, it is currently only available by accident. In December 2002, Andrew D'Addesio discovered that an Edith DLL was distributed with the final version of The Sims Online, called "EA-Land". By performing modifications to the game executable, it was possible to get Edith to start on its own (which is how the above sceeenshots were obtained)[10] . Since EA-Land is a version of The Sims Online, however, it is impossible to get ingame without a server, so *the live debug features have been lost*. Since then, various other versions have been made available, such as one in the early Pre-Alpha version of The Sims Online. Unfortunately, none of these versions can be made publicly available due to copyright concerns, so Edith is currently only accessible by a select few outside Maxis.

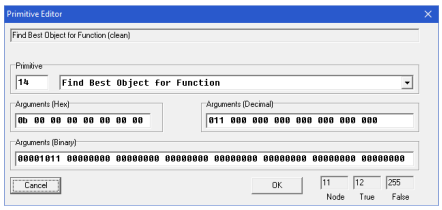
Codex is a Closed Source Freeware editor for objects in The Sims, and is quite unique in that is the only pub-



(a) Main Window, Resource Viewer



(b) A BHAV viewed in Codex.



(c) The "Hex" fallback operand editor.

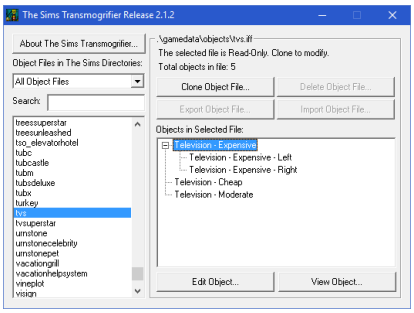
Figure 3.2: Screenshots of the Freeware object editing tool, Codex.

lically available tool that lets users program SimAntics in the intended "Tree Layout" format. The BHAV Editor included with the tool was modelled off of an academic tutorial describing SimAntics taught at Northwestern University[11], and on a series of videos that The Sims programmer Don Hopkins made around the time of the game's release (which are now on YouTube)[12] .

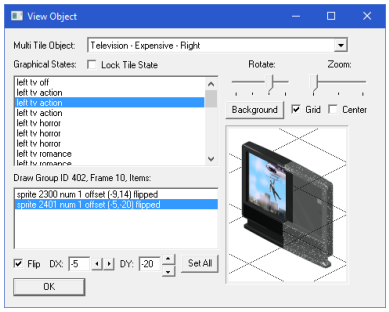
The tool supports all objects in The Sims 1, and even supports saving/reading BHAV tree arrangements via the Edith TREE format, which unfortunately has been stripped from all game objects except from the NPC Maid anyways. As well as BHAV resources, Codex also supports editing Tree Table (TTAB), String (STR#), Entry Points (OBJf), Sound Effect Names (FWAV) and Behavioural Constants (BCON), just like Edith. These editors also attempt to mimic the appearance of Edith's, though the String editor has been improvised.

Unfortunately, Codex only has implements visual operand editors for about 10 primitives, leaving object developers having to resort to editing operand data directly with a hex editor, as shown above. Since Codex does not run a game environment, it suffers from missing many quality of life improvements that Edith provides off the bat. These include smart names for certain Variable Scopes (eg. object attributes), retrieving the names of target object types via GUID and Behavioural Constant names. It is also not possible to edit BHAV resources while the game is running, or debug them via stepping through the scripts primitive by primitive. The Sims must be restarted between each edit, which can get incredibly tedious when the user is attempting to fix bugs with their object scripts.

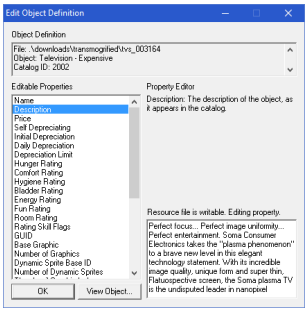
Both of these tools have a similar but important flaw - *it is not possible to re-skin or create entirely new objects with these tools alone*, and the game must be reopened if a new object is created. This is rather unusual, as game object scripts are heavily linked to game graphics, though for Edith this was likely done to separate the content pipeline for the artists from that of the programmers. For Edith, a currently undiscovered external tool was used by artists to create and skin objects, but The Sims object modders used "The Sims Transmogrifier".



(a) The main window



(b) Drawgroup viewing/editing.



(c) Object Definition Editing.

Figure 3.3: Screenshots of The Sims Transmogrifier, a tool for cloning and recolouring objects.

The Sims Transmogrifier is a Closed Source Freeware tool by The Sims developer Don Hopkins, which allows users to clone existing objects, change their graphics, and manually change some resources (like SLOT)

via an exported XML format containing descriptions of the fields they expect. It's important to note that with Transmogrifier, it's not actually possible to create new objects from scratch - only clone them, and processes like modifying or adding extra Drawgroups to an object required manual modification of an XML format. Despite this, the tool is rather easy to use for its purpose of recolouring objects, and provides various facilities to make this easier for the average user such as generating rudimentary z-buffers and far zoom graphics for sprites automatically. It is also includes a (limited) visual editor for Object Definitions (OBJD) and Catalog Strings (CTSS), which lets users change things like the object's price, name, ratings, category and appearance in the catalog.

There are various other tools, such is **"Iff Pencil 2"**, mostly focussed around modifying IFF files without any high level description, where cross references have to be manually worked out and most formats must be modified with a Hex Editor. You may notice that all of these tools are closed source, which limits how useful they are to both future developments in The Sims custom content, and showing how they work behind the scenes to future generations. It's clear that the current solutions are very lacking, which drives the main motivation of this project - making a tool that is both feature complete, standalone, and potentially extensible through its open source nature.

Chapter 4

Planning and Requirements

4.1 General Overview

I have chosen the Waterfall model as the main software design process for this project, mainly because it has worked the best for me personally in previous projects - in both time taken and quality of the result. Before starting work on anything, I drafted a list of high requirements covering the entire scope of the project. Note that the "design $\rightarrow$ implementation" phase of the model, normally performed for the entire system, was executed for each distinct component separately. This decision was made due to the scope of the system, and its potentially evolutionary nature - should any problem be discovered in the implementation phase of any component, it would not require a redesign of any others.

The requirements elicitation for the project, since very few veteran The Sims Modders were available for question, was mostly "archaeological" in nature. I analysed the existing tools extensively; what they did right and wrong on the interface side, how well their designs fit the content system of the game, how these tools were used back in the day, and more specifically, how to make the process easier for an object developer.

To allow for some variation in feature completion, I prioritized each feature, and all sub-features within them using the MoSCoW method, which stands for "Must have, Should have, Could have, and Would like but won't get". This is a popular and effective method of prioritizing requirements in a time constrained environment, and makes it easier for extraneous features to be dropped if they do not meet the deadline. Throughout this section, I will tag specific requirements with letters to specify the group each of these belongs to, [M] being "Must Have", [S] being "Should Have", [C] being "Could Have" and [W] being "Would Like". Generally, I will specify the most important requirements first.

Make and Save changes to file [M]

Making changes is obviously something the IDE needs to be able to do. For pure experimentation purposes, being able to modify the resources in the short term is enough... but for any other purpose it *must* to be possible to save these changes so that they persist across future game sessions, and can be distributed to other players.

- **Changes apply instantly ingame [M]:** One of the main benefits of Edith was that you did not have to restart the game to see changes to objects that you made. This saves a lot of time, and streamlines the debugging process by allowing users to make many small changes to resources and see the effects instantly. When using tools for The Sims 1 such as Codex, Iff Pencil and The Sims Transmogrifier, restarting the game into a household *took up to 1 minute on 2004 computers*, which was a huge turn-off for most potential users. Thus, this feature is *crucial* for using the IDE to be a painless experience.
- **Save Changes to IFF [M]:** There's not much point in being able to make changes that you cannot save, but it's also of interest to save the changes in a format that is usable by the original game and its associated

tools. Since FreeSO features no additional primitives and resource types, all changes *should be backwards compatible* to the original sims engine without causing problems. Objects in The Sims are always stored using the IFF format, so saving back out to this format *must* be possible.

- **Revert Changes to last save [S]:** It's very easy to make changes that are either accidental or don't turn out as you expect, and sometimes they can be substantial enough to demand a large amount of time manually undoing. This is easily dealt with in Codex, where changes are not saved until you explicitly do so, but if changes are instantly and automatically applied to the game, there should be some way to revert them on demand.
- **Patch existing IFF Files using special format [S]:** It's of great interest to change the functionality of existing objects in the game, either to fix bugs, add features or simply tweak values to balance gameplay. Distributing modified versions of these objects will still contain the original data, which could constitute a violation of Maxis's copyrights. To solve this problem, a patch IFF format (or "PIFF") should be defined, which should define a set of changes that make would transform the original IFF into the modified one. This would contain the chunks to be added, and a metadata chunk which specifies data to be removed and added to specific chunks. This patch format could also be used to duplicate objects with small changes, as explained below. Due to being a new format, this save mode would only work in FreeSO, so it should still be possible to save modified files from the original game out to IFF.
- **Clone existing objects using Patch Format [S]:** A very popular use of modding tools for The Sims was simply giving existing, general objects such as chairs and tables a new look, new catalog name/description and new price. Old tools which allowed this simply copied the existing object and allowed the user to make changes to it, though this has legally grey implications due to copyright concerns. If we can assume that everyone has the same base object files, then we can use the patch format to define visual changes to an existing object *that make it a new one, but do not overwrite the existing one.*

Object Browser [M]

To reasonably edit objects within the game environment, it must be possible to browse through them in an easy to use fashion.

- **Search by object name [M]:** Edith provides the ability to search for an object in the game by it's OBJD's name, and filter out "Master" multi-tile objects. Codex does not supply any way of doing this, and instead forces users to find objects in a huge list of filenames. If they are in a "FAR" archive the resources will not even appear in the list, and must be selected manually... This makes it very difficult for users to find what they are looking for - so it is essential to take the same approach as Edith and implement a full object search. For my tool, I would also like to allow searching by filename, and multi-tile group master name. This should be reflected in the results view, which should visually group objects by the file they are sourced from and by their multi-tile groups.
- **Make new Objects [S]:** While the IDE simply modifying existing objects could place it as reasonably "complete", one of the main goals of the IDE is the ability to create entirely new objects from scratch. It should be possible to save these to their own generated IFF resources, and it should also be possible to add new objects to existing files. This requirement is borderline "must-have", but it can be dropped if it is too hard or the project falls behind schedule.
- **Object Thumbnails [W]:** Something that no current tool provides, but would be an obvious improvement for usability is an Object Thumbnail preview alongside the object selection. This would allow users to see, at a glance, what the object looks like ingame before opening an editor window for it. For FreeSO, this would require tie-in to the GPU world rendering libraries, but assuming some framework is developed for this as part of the tool it should be easy to achieve.
- **Search by primitive use, tree name, other internal data [C]:** It could also be useful to search for objects by their content. New users could potentially search for uses of a specific primitive to see what its function is, the contexts it is used in and how it is used. A user could also search for all the trees that could potentially be called by a "Run Named Tree" primitive, for example many objects define a "pickup sound" tree that is run on an object when a sim attempts to pick it up. This could constitute some rather difficult changes to the content system, however, so this feature is likely not worth the effort.

Live Debug of a SimAntics Instance [S]

To achieve a significant improvement over the current publicly available tools, it should be possible to utilize a running SimAntics VM to debug scripts and other resources in any given object - as they are initially or after changes. This kind of feedback will make it much easier to both explore the functionality of existing game objects, and inform changes a user needs to make to fix bugs with an object they are modifying.

- **Object Inspector, to show active objects on lot [S]:** In a given household, it's not entirely obvious which objects are present in the lot, as specifically "controller" type objects tend to be invisible. The view should simply summarize all objects, their IDs, names and what object they contain/are contained within... though if possible, their position in the household should be made clear to the user. It is also important to be able to select these objects from the UI so that we can select them for use via a "Tracer" or simply just looking at its internal state at any given time.
- **Use BHAV Editor as "Tracer" [S]:** Just like Edith, the IDE should be able to hook onto objects at runtime, catch breakpoints placed in the code, and let the user step through scripts primitive by primitive. Ideally, this should also allow the user to **modify and view object state at runtime [C]**, so that the user can see exactly what the object's state is at any given point at time to see how it will affect the execution of a given BHAV. This will be expanded upon in the BHAV Editor section. This is part of the main goal of the editor, so it is important to have, but not necessary for function.
- **Global State modification [W]:** Edith provides some functionality to view and edit Global state, such as the global simulation variables. In The Sims Online, these global simulation variables are largely unused - things like Budget and Zoom Level are now handled either on a per-sim basis or client side - so this feature is not nearly as useful as it was for The Sims 1. Regardless, there are a few variables which are still used, eg. "InHardCoreCity" and "Debug flags", which could be of interest to change.

Linux/Mac Support [W]

Though FreeSO did not yet support these environments at the time the project was started, the possibility was there, thanks to the use of Monogame as its backing engine. Monogame is designed to be cross platform, running on top of the open source "Mono" C# runtime. It supports bindings for Windows, Linux, Mac and even mobile devices such as iOS and Android. Unfortunately, FreeSO's code itself was not cross-platform at the time, but since these requirements were drafted FreeSO has been extended to work cross-platform, with a few minor issues remaining. The most important factor to consider here is that *it should be possible in future* for the IDE to be ported to Linux/Mac using the Mono framework, and *the technical choices that we make should support this being possible*.

Operate without Game Environment (using just Metadata) [W]

Though this feature is a little counter intuitive to the desire for integration with a live game instance, it would be useful for people who wish to edit or view a game object but do not have the game installed and do not wish to download it. To make this possible, the editor would generate metadata about the objects, global scripts, constants, animations and strings. This metadata could be legally distributed with the editor itself, as it would behave similar to a "manifest" - it would only contain a resource listing, not the resources themselves. People would not be able to view the contents of resources they haven't loaded, only know what their names and IDs are, and that they exist. The live debugging facilities of the editor would also be disabled, as it would not be linked to a SimAntics VM.

4.2 BHAV Editor

The BHAV editor is invariably the main focus of this project, so without its presence, the IDE would not be complete. The focus here is not necessarily to vastly change the design up from Edith and Codex, but to streamline it. The editor must use the "Node Graph" representation of BHAVs, as this is the intended and most natural format for them to appear in.

Add/Remove/Change flow of primitives [M]

The most basic functionality required is simple addition, removal and restructuring of existing primitives. This is necessary to make any change to a BHAV, as most functional changes require at least one primitive to be added, or a true/false pointer to be changed. Even without operand editors, it is still possible with just this feature to make very simple changes to BHAVs, but they will be very limited in scope.

Specialized visual Operand editors for each primitive [M]

As discussed, each primitive has a unique set of parameters, which can change details like the target data a primitive uses to its entire function. These operands are all entirely unique, but still share some similar properties, such as accesses to a specific state variable via a "Scope/Data" pair. Thus, it is *very important* to have unique editors for each of these operands, though *it may be possible* to design a framework to make the creation of these editors easier in the long run *by abusing these "similar" parameters*. A Hex-Editor like the one provided by Codex is not sufficient, as it would introduce serious usability concerns.

Display summary of Primitives' Operands on Tree visualisation [M]

On the tree itself, it should be possible to summarize the operand of a primitive on its representation in the node graph. This should display all of the information the operand can provide, in a concise and easy to understand way, so that its function can be inferred at a glance. This summary should also utilize the context of the BHAV in question to provide names for things like object types, local variables and parameters. Without this function, trees would be very difficult to understand without clicking on each primitive and viewing the operands manually.

Group Primitives by function [M]

Edith and Codex have the user select primitives from a very crowded dropdown box, which makes it hard to find the primitive you want unless you know exactly what you are looking for. For my IDE, I would like to experiment with grouping each primitive by a related function, and selecting them from a "palette" which allows the user to select a group and list all of the primitives available in it, for placement in the BHAV. Groups should also be colour coded, to allow users to infer the high level function of a primitive at a glance, and better understand the structure of a program. The inspiration for this is the MIT Lifelong Kindergarten Group's visual programming language, Scratch, which also groups and colours primitive "blocks" by function.[13] This is very easy to implement, so I've listed it as something that must be tried out.

Tracer [S]

As explained above, a "Tracer" mode would allow the user to step through the execution of BHAVs on any live instance of an object in a SimAntics VM. While this feature is active, the user should still be able to perform modifications to the BHAV on demand, as well as view and change the live state of the target object and its thread. This tracer should also show the active stack of the paused thread, allowing the user to see where specific BHAVs were called from.

Position, Comment, Label, and Goto annotations via TREE chunk [C]

Edith allows the user to save out additional information like comments, and the arrangement of primitive to aid future editors understand what a BHAV's function is. Users can also connect primitives to "labels", which can be jumped to using "goto" blocks. These avoid the user having to create large sweeping connections across the node graph to achieve loops or early exits. Unfortunately, this information is contained within a proprietary format, so supporting it could prove to be more difficult than what it is worth at this time.

4.3 Other Resource Editors

Tree Table (TTAB) Editor [M]

Tree Tables define the interactions available to the player when they click on an object, so to do anything other than modify existing interactions, the IDE must be able to edit the TTAB resource.

- **Add/Remove/Modify/Reorder interactions [M]:** To make substantial changes to a tree table, it must be possible to modify the interactions within it, as well as add new ones.
- **Automatic edit of TTAs resource [S]:** Each interaction in a Tree Table has a corresponding entry in a TTAs string resource, which defines the name of the interaction in the game in all possible languages. The editor should be able to generate and modify these resources automatically. It would be nice to allow the user to edit the strings for each language individually, as well.

Object Overview with Definition Edit [S]

For existing tools, objects are treated more like resource collections than important entities defined by select resources within their IFF file. It is important to reflect this in our "Object Window" design, which should keep a currently selected "object" in mind as context for all resource editors to consider.

- **IFF Resource Browser [M]:** It is necessary for the user to be able to browse through and edit the resources that an object uses. This functionality should be present on the object window itself, and utilize a variety of unique editors to allow viewing and modification of resources, categorised by type.
- **IFF Window for Global/Semi-global Resources [M]:** Each object in the game references a Global IFF file, and can reference a Semi-Global IFF file, as explained in the Primer. These IFFs *do not contain any objects*, so the resource editor must be available *in a standalone fashion* to allow these to be modified.
- **Select main resources for selected object [S]:** Certain resources in an object are flagged as its main resource of that type. For example, an object defines a specific TTAB as the one it uses for its interactions. Existing editors would have you set these manually by ID, but it should be possible to select them directly from the resources view, and see object relevant resources at a glance on the resource list.
- **Modify/Edit Multi-tile Object grouping [C]:** Many objects in The Sims are actually multi-tile objects, which are made up of many single-tile objects, as explained in the primer. Edith and Codex require the user to manually edit all of the cross references to make this possible, which is quite hard for a novice user to understand. The IDE should allow the user to view all the objects in a multi-tile group, change an object's multi-tile group, and make new ones. This may prove quite difficult to implement, and it is not necessary to make most types of objects.

String (STR#/CTSS) Editor [S]

Strings are used for a variety of functions within the game, from defining the animations that an object's scripts use, to what appears in dialogs that it spawns. It is not necessary to be able to modify these resources to substantially change an object, but they are simple enough that the editor should support them.

- **Add/Remove/Edit/Reorder Strings [M]:** For a string editor to function, it must not only be possible to modify strings, but also to add, remove and re-order them. For example, a user must add a new string entry to a STR# resource to include a new animation for use in the scripts, or to add a new dialog for the object to pop-up.
- **Support alternative "Language Sets" [C]:** String resources in The Sims have been designed to support multiple languages, via defining "Language Sets" to use for a number of pre-defined languages. It could be possible to edit strings for all of these unique language sets, allowing users to translate existing game objects, or include multi-language support for their newly created objects.

Graphic (DGRP/SPR2) Editor [C]

Being able to modify object graphics is not one of the main focuses of this project, but it is required in at least a basic sense to allow the user to create an entirely new object from scratch. It may be quite hard to implement, however, so if it does not make the deadline users could potentially use Transmogrifier to make up for the missing functionality.

- **Replace existing Sprites/Palettes with Export/Import [M]:** Just like the functionality Transmogrifier provides, it should be possible to export an object's sprites, modify them, then import them back in. This would allow the user to replace object graphics with entirely new ones, (as well as make new ones for a Drawgroup editor to use) so if possible within the time-frame it is a rather important feature.
- **Modify and add Drawgroups [C]:** Though it's possible to modify existing object graphics by simply changing sprites, you cannot add new graphical states or change their layouts without a Drawgroup Editor. This is not necessary for modifying existing objects, but is required in at least a basic sense to allow the user to create entirely new objects. A more complicated editor is not necessary, but would be nice to have.

4.4 Technical Choices

4.4.1 Language, libraries and platform

Since the IDE will tie directly into FreeSO, the re-implementation of the engine used in The Sims Online, it makes the most sense to write the code in C# so that it can directly utilize all of the libraries and functionality provided by the original game engine. Without these, a large portion of handling code would have to be rewritten, specifically for all of the file formats, and interfacing with the game environment would require a complicated messaging system with serialization for all VM state objects. This could potentially take as much time as the project itself took, so this option was off the table.

While FreeSO provides readers for all of the resource files in The Sims Online, it does not support writing them back out, so the engine and libraries will still have to be modified to support this. Also, though it is possible for both FreeSO and the IDE under development to support Linux and Mac through Mono, no effort was made at the time to compile FreeSOs main client for anything but the WindowsGL Monogame target. Due to this being largely unrelated to the task at hand, the IDE will *probably only function on Windows* for the duration of the project.

4.4.2 Interface with FreeSO

To access the existing game content for modification, we can hook into the FreeSO content system directly, which manages references to all content in the game to access it on demand. By reading from these references, we can retrieve a searchable list of any kind of resource - objects, animations, anything the game accesses itself. By writing to the references while the game is active, we can dynamically remove or add new resources to the game environment without needing a restart.

One option for achieving this would be developing the IDE entirely separately from the game itself, and having it interface with the game through some form of "Plugin" style message passing interface. Unfortunately, due to the scope of accesses the IDE might have to perform on the game, an incredibly large number of unique messages would have to be implemented. Making these as needed instead of in advance may have negative implications for the final design of the IDE, and not enough information is known about the desired final design to scope them out in advance.

Instead of this, I decided to develop the IDE directly on top of the game engine, as I am already familiar with its code. The IDE executable simply starts FreeSO and injects an instance of itself that the game can call back on when a breakpoint is hit, lot is entered, or another event we are interested in occurs. It should be able to access all resources and state used by the game, as it is "below" the IDE in the inheritance tree - all public classes can be utilised.

4.4.3 UI Framework

Since the function of the IDE depends entirely on the GUI it supplies to the user, the choice of which UI framework to use is crucial. When considering this, I needed to take into mind the presence of complex interactive UI such as the BHAV node graph view, how lack of a "familiar look and feel" would impact the user, and the future desire of making the application cross platform. Using C# and Monogame, there are a number of choices - so it is important to weigh the pros and cons of each before we make a decision.

Windows Forms

Windows Forms is the legacy GUI class library included as part of the Microsoft .NET Framework. It allows .NET applications to present the user with Graphical User Interfaces using the Windows look and feel. In Windows, it renders using the Graphics Device Interface (GDI), and is translated to native calls when run in Linux/Mac via Mono.

+ Cross platform using Mono, an open source implementation of the .NET framework, which allows C# .NET applications compiled for Windows to run on Linux and Mac.
+ Lets the user design applications which fit with the native Windows look and feel, easing comprehension and work-flow for Windows users.
+ Many powerful components to build the UI on top of, many similar to those used already by both Edith and Codex, with the possibility of making custom components - which would become useful when designing a framework to build operand editors out of generic pieces for the BHAV editor.
+ Very easy to use; Visual Studio comes with a visual editor for Windows Forms.
+ No graphics acceleration hardware is required.

- On its own, a software renderer would need to be written for any embedding of ingame graphics such as Drawgroups, thumbnails and animation previews. This would involve a lot of extra work which could mostly be considered "duplicate code" anyways, introducing future disparity if it needs to be changed.
- Because Windows Forms uses GDI (which is only partially hardware accelerated), it is incredibly unsuitable for complex animations and large scale free-form component drawing a SimAntics tree viewer would require.
- Cross platform support is limited in places.

Windows Presentation Foundation (WPF)

WPF is the intended successor to Windows Forms, running on DirectX rather than GDI for UI, with the intention of supporting hardware acceleration to allow high performance, dynamic user interfaces. It was introduced with Windows Vista.

+ More powerful than Windows Forms, thanks to hardware accelerated components and support for vector graphics.
+ Supports hardware acceleration via DirectX, so offers much greater performance for interactive UI components than Windows Forms, but it is not required.

- *WPF is NOT cross platform*, as Mono does not support it, and has no plans to in the near future. While this is not a problem for the scope of this project, it will become a problem if the IDE is ever ported to Linux *in future*.
- Graphics acceleration is not required, but without it performance will take a severe hit.

Monogame

One option is to build the UI for the IDE entirely in Monogame, similar to FreeSO's GUI. This would essentially be building the UI on top of a game engine - which would allow for very powerful custom elements, but would require a lot more boilerplate code to string together.

+ The code editor could benefit greatly, as it relies on completely graphical display. Zoom, smooth animated scrolling and helpful animations would be simple and effective for portraying things like flow of execution.
+ Object and animation previews can be made possible through existing code in the FreeSO code base, which is made to run on Monogame.
+ Cross platform, but needs some fiddling to compile for Linux/Mac.

- All UI components that users take for granted in Windows would have to be rewritten from scratch, which could prove to be incredibly complicated for even simple UI elements like text entry fields or ComboBoxes.
- Even when recreated, the UI controls would feel incredibly out of place and unintuitive compared to using the native controls users would find in other Windows applications. This could severely impact the user experience unless done very carefully.
- Requires graphical acceleration, but since FreeSO runs entirely on Monogame they would need some form of 3D acceleration to run and debug their objects in the game environment anyways.

WinForms/WPF + Monogame Hybrid

An interesting choice for the UI is one that combines the best of both the .NET UI frameworks and 3D accelerated components. A dedicated GUI system could be used in combination with specialized Monogame components for the places that need it most, such as Object/Animation previews and the SimAntics code view.

+ All the benefits of Monogame AND native solutions, using both for places where they fit best.

- Despite not requiring graphical acceleration for most parts of the UI, the requirement to have it at all for any crucial parts, such as the BHAV editor, will lock out users without graphics acceleration.
- The box and arrows renderer could still prove hard to set up, though it would likely also be difficult with Windows Forms and WPF as it would require custom controls with special painting.

In the end, I went with the Windows Forms/Monogame Hybrid. The main reason for this were that the Object and Sim rendering code could be reused, and that I believe the BHAV tree view would be more natural to program in a game engine compared to a traditional UI framework. The reason Windows Forms was used instead of WPF was solely for future Mac and Linux support, since Mono does not support WPF for those platforms. This decision did not come without its implementation problems, however, as will be explained in the Backend chapter.

Chapter 5

Backend

5.1 Main program and Injection into FreeSO

One of the first things the program needs to do is boot FreeSO in a way that it can call back to the program on certain events, such as when a new household is opened or when a breakpoint is hit by the SimAntics VM. To do this, I decided to use a pattern called **"Dependency Injection"**, where an interface for external communication has been defined in the client ("IDEInjector"), and an IDE class "IDETester" which implements that interface has been "Injected" into a static variable before the program has started. This static variable can then be accessed by FreeSO using the defined interface, which contains all the events it needs to fire - specifically:

```
public void StartIDE(VM vm) { ... }
public void IDEOpenBHAV(BHAV targetBhav, GameObject targetObj) { ... }
public void IDEBreakpointHit(VM vm, VMEntity targetEnt) { ... }
```

Note that this interface also includes a facility to open BHAVs from FreeSO - as I have made it possible to open any interaction's action tree by shift-clicking it in the pie menu. The StartIDE function is run when a household is entered, and starts Volcanic's main window in another thread with the active VM as a debug target.

Windows like BHAV Tracers, Object Windows, Iff Resource Windows and BHAV Editors are all tracked by "Window Managers" central to this main window. This is mainly to prevent two windows being opened on the same resource, which could result in a confusing user experience. When the IDE attempts to open a new window of these kinds, it does it through the classes **"IffEditManager"** and **"BHAVEditManager"**, which check if the requested resource already has a window open for it, and brings the focus to it. This is especially useful for breakpoints, which attempt to open a new BHAV Tracer to debug the entity that hit the breakpoint. If a Tracer is already open, it is simply notified that the thread has paused again and is brought to the front.

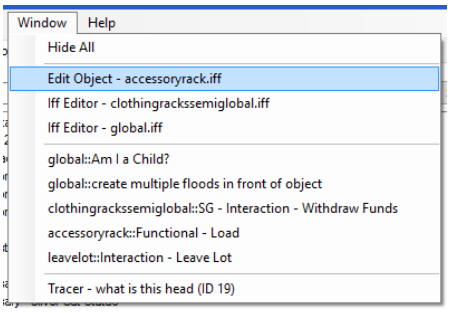


Figure 5.1: Window Manager UI

Using this information, the main window also provides the user a facility to view all of the currently open windows in the IDE. Clicking on any window will bring it to the front, and the user is also able to minimize all windows to reduce clutter.

5.2 UIExternalContainers

As decided in the requirements section, certain UI elements would have to be rendered by Monogame, such as the BHAV tree view and any object thumbnails. However, as some initial prototyping revealed (which will be discussed extensively in the next chapter), attempting to render these elements and the game concurrently caused a large number of complications.

The solution was to have the game run these elements *entirely within itself*. The IDE would manage registration of **"UIExternalContainers"** for any elements that need Monogame handling, which defers their update and render steps to the game itself. These allow external use of the FreeSO UI system; they take in any input provided by an external source, run updates *between game execution on the UI thread*, and signals an event when a new frame has been rendered. Frames are rendered to an internal Render Target, which is read from and converted to raw ARGB format when sent to listening UI elements as an event. The size of this render target can also be custom defined and changed at runtime, at which point the Render Target is regenerated with the same size and a redraw is forced. UIExternalContainers also "lock" themselves during any processing, so that any external accesses (which should also attempt to "lock" it) cannot occur at the same time.

UIExternalContainers allow the user to utilize the entirety of the FreeSO UI system, including standard controls and features provided by the FSO.Client.UI namespace. These include labels, wrapped typesetting, images, animated buttons, and access to the nested container system, which allows UI components to contain multiple others for easier positioning of their elements. The practice employed by the IDE is to subclass the regular UIContainer class, implement the UI as desired and add it to the UIExternalContainer. These subclasses are contained entirely within the FSO.IDE namespace, so do not clutter up the client code.

A standard control was developed to allow this framework to be used with Windows Forms, dubbed the **"FSOUIControl"**, which manages a UIExternalContainer within itself. This control performs simple tracking and forwarding of mouse input, keeps track of and forwards any changes to its size, and draws out frames received from the UIExternalContainer. This control can be placed in any Windows Form, and has no restrictions to its position or its size - which can be changed at any time. This allows for the treatment of Monogame handled UI elements as intuitive embedded controls within Windows Forms, as desired.

There are a few drawbacks to this approach, however. Since the data has to be copied from the Render Target on the graphics card back to the CPU, then passed through as a Windows Forms bitmap, processing the frames can become *incredibly CPU intensive at higher resolutions*. To alleviate this problem, UIExternalContainers do not redraw until they have been explicitly told to, using the ExternalDraw flag in the UI engine's shared data section. (eg. `state.SharedData["ExternalDraw"] = true;`) This is a bit of a hack - but implementing a more complete system across the whole UI could have proven to be quite a large task. Performance issues are still noticeable when scrolling with large BHAV tree views (eg. at 1080p), so in the future it might be important to optimize this process further.

5.3 ResActions

Since the IDE is on another thread from the game, it is important to have a way of modifying the game content without causing any race conditions. Another concern is that operations which utilise the graphics card, such as object sprite updates, *must be performed on the game thread*. It's also important to have a standard way of making changes to arbitrary IFF chunks so that we can track them for display, save and potential reverts.

To solve all of these problems, I developed a simple system built around "ResActions" - actions with the intention to modify the contents of an IFF chunk that can be queued and run by the game thread later. The user simply makes one of these with the action of their choice, as well as optionally specifying the chunk it changes,

and sends it to the game thread for later execution. This also notifies the Change Manager that the specified chunk has been changed, as will be discussed later.

Here is an example of the system being used, from the STR# resource editor:

```
var str = StringBox.Text;
var ind = SelectedStringInd;
Content.Content.Get().Changes.BlockingResMod(new ResAction(() =>
{
        ActiveString.SetString(ind, str, ActiveLanguage);
}, ActiveString));
```

Thanks to the anonymous function, we can define unique changes inline without much effort. Note that we are using the **"BlockingResMod"** variant, which waits until the game has completed the operation before continuing. There is another variant, **"QueueResMod"**, that does not block execution for the caller thread.

This system is not yet perfect, however. Currently, if the game itself performs any kind of modification (usually just on resource load), a race condition could occur if the UI is reading the chunk at the same time, since reads do not use this system. This problem could be solved via extensive locking throughout the code, but this was deemed too extreme a task to perform within the timeframe.

5.4 The PIFF Format

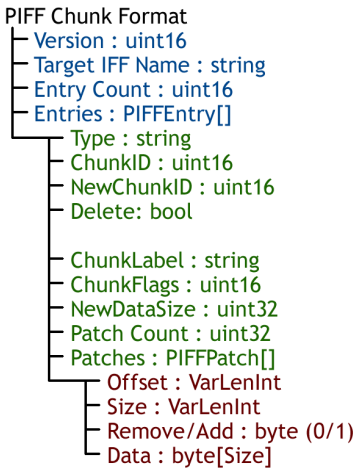


Figure 5.2: PIFF Chunk Format

As discussed in the requirements section, there is a desire for a "Patch IFF" format, or "PIFF", that allows the user to save patches to original game IFFs. These patches could be distributed to other users without having to redistribute any of the original IFF contents, avoiding potential copyright violations. Since the patches are applied only at runtime, they would also leave the original game files completely unmodified, meaning that the user could delete simply them to return an IFF to its initial state.

To get an idea of how this kind of format worked on a technical level, I looked up the "IPS" format, commonly used to distribute ROM hacks for video games. I gathered that their format simply stores a list of replacement changes to an existing file, which are just applied directly. Since it has been designed with ROM modification in mind, where statically linked binary code with many cross references are contained, there is no facility for inserting or removing data within the file in a list like fashion, and the result is expected to be the same file size. For the purposes of the IDE, where most changes would add or remove data from a file (eg. strings and scripts), this would not do.

Instead, I decided upon a custom format which would contain additions and deletions to various chunks that could be applied to an existing IFF. This decision was inspired by the material taught in Algorithmics I & II, and even uses the "Longest Common Subsequence" dynamic programming algorithm to calculate the areas where data needs to be removed or inserted in an optimal fashion. However, since the algorithm has $O(mn)$ time and space complexity, I decided to run this algorithm on each 1024 byte chunk of the data separately. This means that the result will not be optimal, but in most cases it will be "good enough" to track moderate changes. Using memoization, it is possible to find the list of additions and deletions without generating the full dynamic programming table, but this cannot be avoided in the worst case. It is also possible to generate the LCS using Hirschberg's algorithm, which has linear space complexity, but that is an exercise for the future.

The PIFF format is simply an IFF that contains all of the chunks to add to the target IFF file, and a custom "PIFF chunk" which contains the name of the target IFF, and an array of changes to its existing chunks. Each of these contains the Type and ID of the chunk to replace, as well as a new ID, name, data size and flags for it. It also contains the list of additions and deletions to apply to the data to get to the target file. It's also possible for chunks to be entirely deleted, in which case no changes are specified.

FreeSO has been modified to load these PIFFs as the game starts, and stores in a "PIFF registry", indexed by target file. When another IFF is loaded, it checks its name against the registry, and applies any patches that have been loaded for it. IFFs remember their initial state and what patches have been applied to them, so it is possible to revert an IFF to its last saved patched state by simply reverting the data and re-applying the patches in order.

5.5 Change Manager

One of the concerns raised in the requirements section was the ability to keep track of changes to an IFF file, in a way that allows them to be viewed, saved and reverted. To achieve this, a class was added to the FreeSO Content system to track changes made to any chunks and IFFs, called **"ChangeManager"**. This class tracks a set of registered changes to IFFs, and handles the queue of ResActions that the game needs to execute next frame. To register a change to a chunk, all a code segment has to do is call ChunkChanged on the manager with the chunk as a parameter.

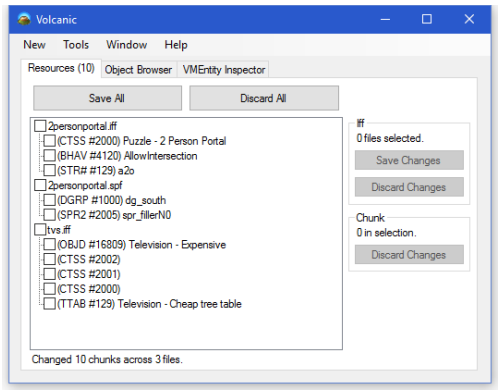


Figure 5.3: Volcanic Change Manager UI

The ResAction construct has been designed to do this automatically, but it is possible to perform manually nevertheless. This flags the IffChunk as changed via its RuntimeInfo field, and runs IffChanged to add it to the collection of IFFs for which a change has been registered.

The UI retrieves this in a thread safe manner every time the window regains focus, running through the chunks of each IFF to see which have been flagged as modified, and displaying them on the TreeView. On this view, the user can select any combination of Chunks and IFF files to save or discard changes to. As discussed above, when a change is discarded, it is reverted to its saved initial state, then any previously saved patches are rerun. However, when a chunk is saved, depending on where the IFF is from, a number of things can occur. If the IFF is a read-only iff included with the game, it is saved out to the patch format, using a PIFFEncoder class to write out patches for any chunks with data different from the initial read-only state. However, if the change is to an object loaded from the Content/Objects folder, it is saved as a normal IFF and overwrites the old file.

Chapter 6

BHAV Editor

6.1 Prototype

Before starting on the BHAV editor, I decided to build a prototype to test a number of things. First, and perhaps most importantly, the planned use of Monogame to handle input for and render certain elements within the UI. I also wanted to test out the possibility of automatically generating tree layouts from the primitives within a behaviour like Codex does, as most behaviours in The Sims Online have had their positioning data stripped. For the layout concern, the prototype demonstrated that this was no problem, however I quickly ran into a large number of blocking issues when trying to get Monogame both embedded in the UI at all, and working alongside FreeSO.

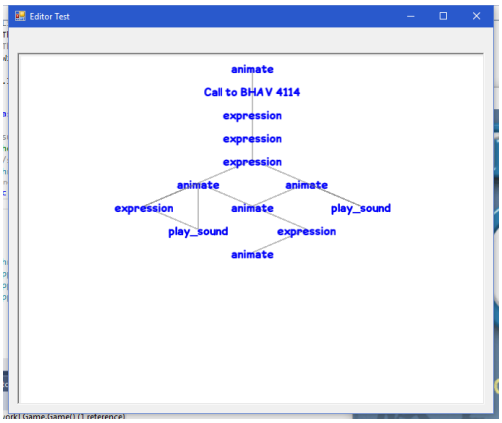


Figure 6.1: Early Prototype BHAV Viewer

I very quickly discovered that Monogame was not designed to work over multiple threads. Without modification, the Monogame library blocks programs from spawning a new graphics device from a thread other than the thread owning the main Game object. Monogame has been designed expecting the user to only use one Game object per application, so creating another was simply not an option! Even then, attempting to circumvent this by using the game's graphics device caused extreme flickering and undefined behaviour - as the render processes of each thread were interfering with each other. Modifying Monogame itself to allow creation of graphics devices on external threads caused another issue where the game's render window ceased functioning entirely.

Despite these issues, the prototype demonstrated that it was easy to access the FreeSO content system from an external thread without issue. Since this was the first task I performed, this laid the groundwork and confirmation that the planned features like the ResActions, Change Manager and even just content accesses throughout the code were possible. After this failed prototype, I began work on another which focused around building the UIExternalContainer system to circumvent the Monogame issues, which eventually developed into the final product present in Volcanic right now.

6.2 Final Product

The final version of the BHAV Editor has met all of the requirements, and arguably the expectations that I set, and is definitely the most impressive and complete part of Volcanic as a whole. During the evaluation, many

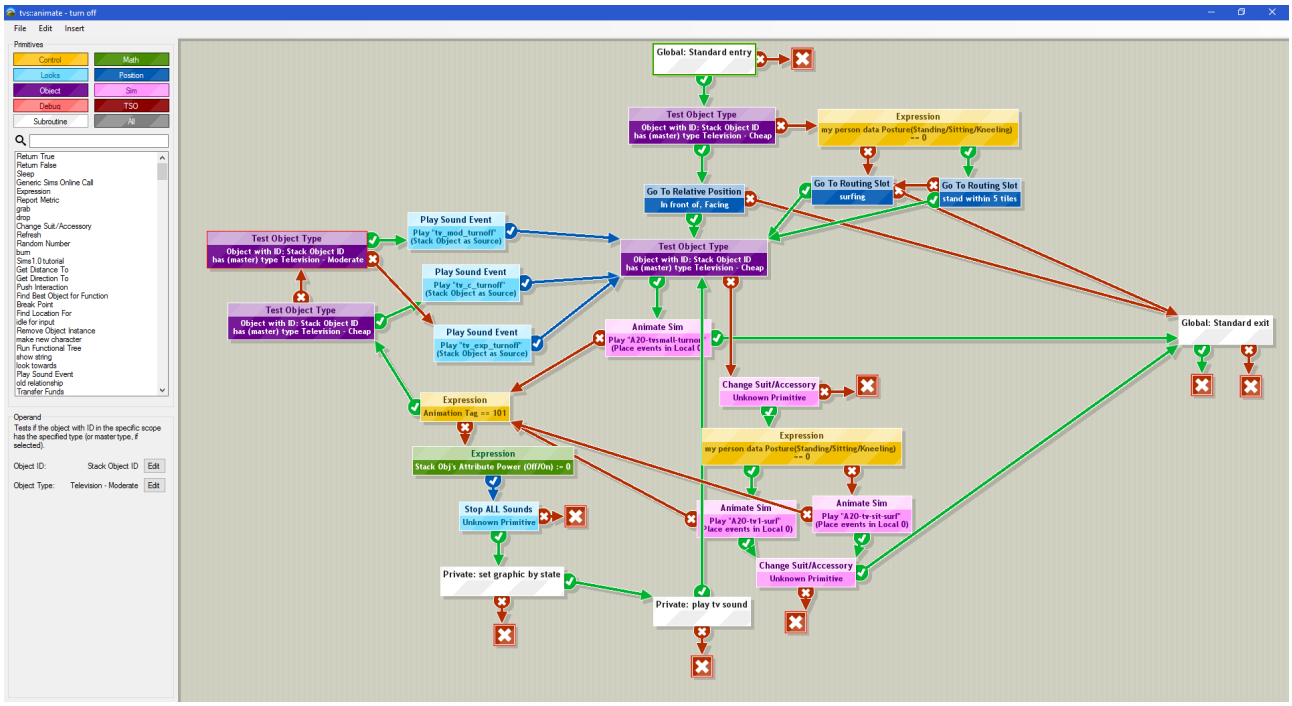


Figure 6.2: A complex BHAV in the final version of Volcanic's BHAV Editor

users said that they were blown away by both the visual style and technical design of the BHAV editor, much more than the other components.

Firstly, the tree view has undergone a facelift. It uses a custom UIExternalConatainer, the **UIBHAVEditor**, to provide a graphically and interactively complex interface using FreeSO's UI framework. The background is now patterned for easier comprehension when scrolling and uses a darker, slightly orange shade of grey to make it stand out from the surrounding UI. Primitives themselves are much cleaner - they now clearly differentiate their title from their body, dynamically resize to fit in tighter spaces depending on the contents of their body, and are neatly coloured depending on what primitive group they are part of. A primitive's true/false/done branch nodes are no longer anchored to the bottom, and instead point out in the direction of their destination primitive. The colours are deliberately easy on the eyes (slightly muted), but also very noticeably distinct to make their meaning much easier to glance. Finally, everything has a slight solid shadow to make it stand out against the background.

To scroll, you simply drag the background with a left click. To move primitives, you left click on their body and drag to the desired position. To change where an exit node points, the user drags from it to the new destination primitive, or empty space to clear it. When a primitive is left clicked, it is selected, and a specifically tailored operand editor for it will appear on the bottom half left sidebar. Changes made using these operand editors will apply immediately, and update the display on the tree view accordingly. Any selected primitive can be deleted by pressing DELETE, which will clear pointers from other primitives heading towards it. Any primitive can be set as the first primitive by using the Ctrl-1 shortcut, or the option in the "Edit" menu on the top bar.

Instead of being placed from a $\tilde{7}0$ item combobox, primitives and subroutines are now placed from a categorised palette, which also lets the user search for what they want by name. When the primitive is selected from the results list, an animated dashed line appears around the tree view, and the primitive will follow the user's mouse over the tree view. It can be placed by clicking anywhere on the tree, though placement can be cancelled by pressing ESC instead. When a primitive is placed, its operand value is set to the default value, and should be configured by the user via the operand editor.

The user can undo and redo any changes they make to the primitive using the normal short-cuts (Ctrl-Z, Ctrl-R), with currently infinite depth. The technical implications behind this will be explained in the BHAV-

Commands section. Finally, the user can place Return True and Return False boxes in the view either by pressing Ctrl-T/Ctrl-F, or from the primitive palette. The rest of this chapter will explain the important technical aspects behind all of these features, and how they were achieved, and what improvements to the design they allowed.

6.3 Editor Scope

Each behaviour in the SimAntics Environment can access resources and state from a variety of locations. Resources are generally from its owner (Private) IFF file, but can also be fetched from its Global or Semi-global IFFs when applicable. Most variable names are the same throughout all behaviours, such as names for the Object Data and Person Data scopes. However, some depend on the target object, like Attributes, and some depend on the active behaviour itself, such as Parameters and Local Variables. To keep track of all the data accessible for a given behaviour in a way that can be utilised by all areas of the BHAV editor, I defined a centralized construct to keep track of these resources and manage access to them, called the **"Editor Scope"**.

The most common use of this facility is returning names for specific variable scope-data combination references, which can be fetched either individually by data value, or listed out as a set of all possible values of a scope, with each given a name. A variable reference's "scope" field refers to where it should come from (eg. my motives, stack object's attributes), and its "data" field generally indexes which variable we should access from that scope. The most common use case for primitive operand display on the tree view is to fetch a single name based on the scope and data value it provides, and simply print that in its text body.

```
public string GetVarName(VMVariableScope scope, short data)
```

The above generally prints a name in the format "Scope Data", for example "my object data allowed height flags", "my object data graphic" or "my person data CreativitySkill". However, for selection of scope data for operand editors, or a full summary of all scope names to be used by the tracer's variable display, we want to fetch a list containing all of the valid "data" values, and names for them. For this purpose, we can use:

```
public List<ScopeDataDefinition> GetVarScopeDataNames(VMVariableScope scope)
```

Which returns a list of "ScopeDataDefinition" structs, which simply contain each data value's name, value, and a description of its purpose. This description is not yet set or retrieved, but the idea is that it could be displayed by UI elements for easier comprehension of a specific variable's purpose in future. Of course, it's also of interest to get resources relevant to the BHAV from the Private, Semi-Global or Global scopes, either for a list of all resources of a type, or for a specific one to get its name or some internal data. There are a few subroutines for doing this for different purposes:

```
public T GetResource<T>(ushort id, ScopeSource source)
public List<T> GetAllResource<T>(ScopeSource source)
public ScopeSource GetScopeFromID(ushort id)
```

These are pretty self explanatory. As far as uses go, GetResource is mostly used for fetching a resource to display information about it, while GetAllResource is used to list all the resources available from a specified scope, generally for operand editors to utilize. When retrieving BHAVs, the last function is used to determine its source scope from its ID range. You'll see how these are used in the next sections.

6.4 BHAVCommands

When editing a BHAV, the user can perform a number of unique operations. These include: changing the true/false pointer of a primitive, changing a primitive's operand data, adding a primitive, removing a primitive,

setting a primitive as the entry point of a BHAV, and toggling a breakpoint on a specific primitive. These operations should ideally perform on the game thread between SimAntics ticks to prevent interfering with any code that is currently running. It's also of interest to be able to undo/redo these operations at will, especially if the user makes a mistake.

With this in mind, I decided to use the Command Pattern to model each of these operations into unique **"BHAVCommands"** which can be sent to the UIBHAVEditor, queued, and executed by the game thread. Each of these commands extends the abstract class BHAVCommand, which simply exposes the functions Execute and Undo.

```
public abstract class BHAVCommand
{
    public abstract void Execute(BHAV bhav, UIBHAVEditor editor);
    public abstract void Undo(BHAV bhav, UIBHAVEditor editor);
}
// in UIBHAVEditor
public void QueueCommand(BHAVCommand cmd) { ... }
```

When a command is executed, it is added to the end of a "History" list, so that its "Undo" action can be run some point in the future to revert the change made by "Execute". Each action generally also manually calls ChunkChanged on the content system, as they do not utilise ResActions (these were created before ResActions existed).

6.5 Primitive Descriptors

There are 68 unique primitives in The Sims Online. The problem is, we need to have custom rendering and operand editors for all of them! Simply designing unique interfaces for these could prove very time consuming, result in inconsistencies throughout each operand editor, and involve a lot of boilerplate code - unnecessarily complicating the codebase. Displaying them on the BHAV tree view is not as much of a problem, but a clear interface still has to be defined to provide their appearances and summaries to the Monogame UI for display.

For this purpose, I have defined an abstract class called the **"PrimitiveDescriptor"**, which unique handlers for each primitive type should subclass and implement. Each primitive in the tree view should have one instance of the type relevant PrimitiveDescriptor, with the Operand parameter set to a copy of its real operand. PrimitiveDescriptors implement a number of functions, which define how various parts of the editor UI handle a primitive:

```
public abstract PrimitiveGroup Group { get; }
public abstract PrimitiveReturnTypes Returns { get; } // true/false or done
public abstract Type OperandType { get; }
public virtual string GetTitle(EditorScope scope);
public abstract string GetBody(EditorScope scope);
public virtual void PopulateOperandView(BHAVEditor master, EditorScope escope,
                                        TableLayoutPanel panel)
```

For most of these, the current Editor Scope is also passed in to provide contextual information the primitive's display may require, such as variable names. Their results directly affect the appearance and behaviour of the primitive in the tree - Group defines their colour, Returns defines which pointers come out of them, and the Title and Body getters define their text summary. The Group and Returns getters are abstract functions so that certain primitives can dynamically change their type or what they return based on their operand. For example

some expression primitives are considered “Math” and return “Done” (x += 2, y := 7), and some are considered “control” and can return “True/False” (x > 3, y Has Flag x), depending on the operator specified in the operand.

The other function, and arguably the most important, PopulateOperandView, fills a given TableLayoutPanel on the Windows Forms UI a with a set of “OperandForms” which provide a visual editor for that primitive’s operand. These will be explained in the next section. At the time of writing, there are 28/68 unique primitive descriptors implemented, each with all of these functions fully implemented and registered using a “PrimitiveRegistry” - the ones that are not are mostly unused or have complex operands that are not fully understood by FreeSO. Regardless, the state of support is definitely much better than Codex, and can quickly approach Edith in the near future thanks to the existing groundwork.

6.6 Operand Forms

6.6.1 Overview

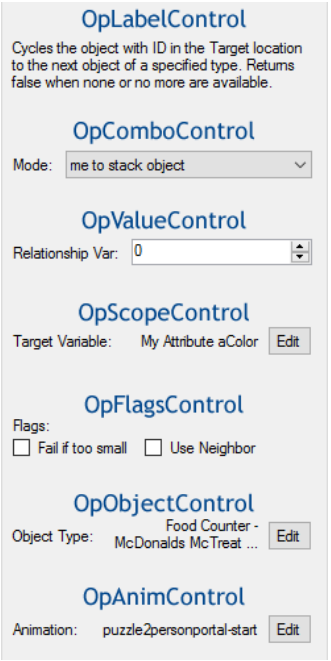


Figure 6.3: All OperandForms implemented and used so far.

As discussed above, creating individual GUIs for all 68 operands would prove incredibly tedious, risk introducing design/functionality inconsistencies, and require the duplication of boilerplate code across all areas. Within a time constrained environment, like this project is, it would simply not be feasible to create even 10 of these editors for the BHAV editor alone. To solve this problem, I came up with the idea of **“OperandForms”**; unique, self managing building blocks that operand editors are built out of.

OperandForms are typically passed a number of things: the BHAVEditor window, the Editor Scope, the operand of the primitive in question, any strings that appear on its UI, any DataProviders (explained later), and the code name of the properties that the form should be modifying. OperandForms read and modify their target properties by name from the operand using C# reflection - literally querying the operand’s C# type to find a specific named property, then accessing it in the target operand at runtime. A small static utility class has been created for this function, called OpUtils, which automatically does any necessary type conversions before storage and abstracts away some of the boilerplate code.

OperandForms are driven mostly by **DataProviders**; classes provided by the PrimitiveDescriptor which define the data values that the operand they are modifying can take, and how they should appear in the UI. The PrimitiveDescriptor can provide premade “static” variations of these DataProviders for data values which are known in advance. It can also implement its own variant of a DataProvider for more dynamic retrieval of data, for example, if the options available for an operand change depending on the value in another field of the operand.

For example, if an operand could take “get/set/adjust” as values for one of its fields, with the values “0/1/2”, it would be able to use an OpComboControl with an OpStaticNamedPropertyProvider created with those names and values. In another example, “Snap to Slot”, the source of the operand’s “Slot Index” field changes depending on its “Source” field, so for “Slot Index”’s DataProvider we would subclass OpNamedPropertyProvider and add some custom functionality to return the correct set of Slot Name+ID pairs depending on the “Source” field. This provider would be given to an OpComboControl, which will let the user select slots from the active source using a combo box.

Currently, there are only 3 categories of these DataProviders, meant for specific OperandForms - OpTextProvider

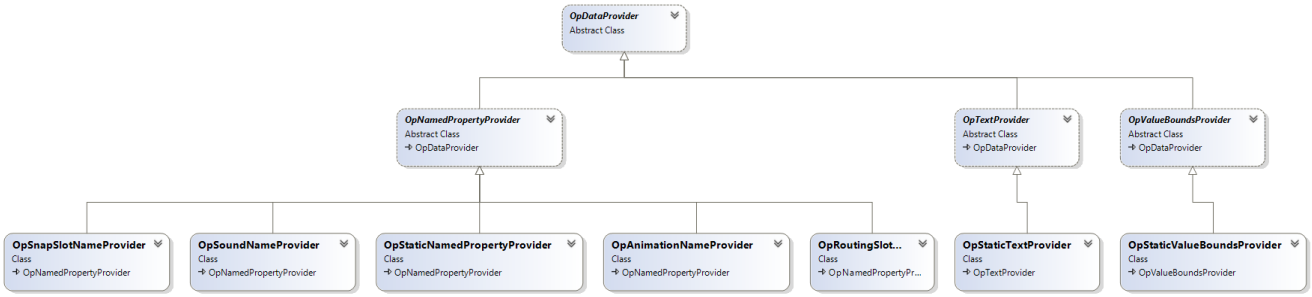
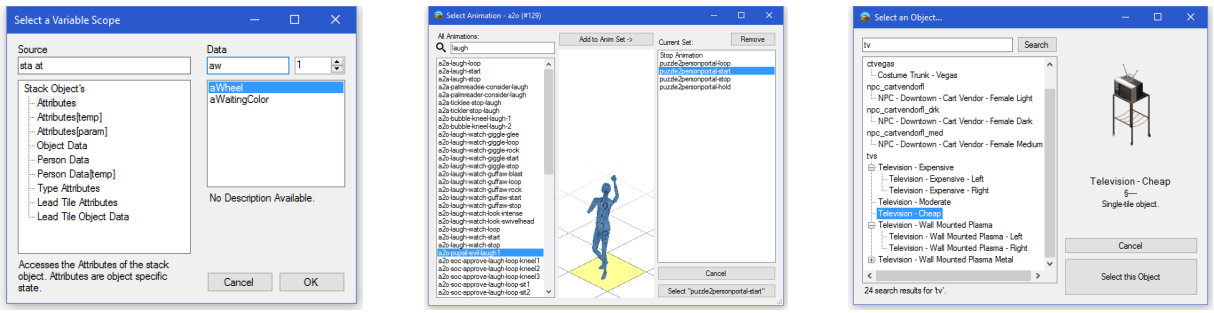


Figure 6.4: All of the "data provider" classes in Volcanic.

for OpTextControl, OpNamedPropertyProvider for OpComboControl, and OpValueControl. The other OperandForms only use data which is in one format/place, so they do not use OperandForms.

One possible alternative to this system which could involve even less workload would be instead using a Windows Forms PropertyGrid to edit operands... but these would not permit the same degree of specifically tailored design, and might introduce a whole other set of problems for implementation. The inclusion of this system has definitely lightened the workload - I'd say that without PrimitiveDescriptors and OperandForms, there would be a great deal fewer operand editors in the final product, and primitive display in the tree view would be in a much worsened state.



(a) Variable Scope Selector (b) Animation Selector (c) Object Type Selector

Figure 6.5: Various "Selector" dialogs which specific OperandForms delegate to.

6.6.2 Example: Variable Scope Selector

Through Operand Forms, it is possible to delegate more complex editing interfaces to external popup windows. These popups could simply be supplied with the same Editor Scope and Operand the OperandForm received, and provide back a changed value on confirmation. It quickly became clear that the common Scope+Data variable references demanded such a complex editing interface, so that variables could be discovered easily.

On the operand editor itself, the name of the variable referenced is displayed, with a button beside it which opens the special selector dialog. This dialog simply shows a categorised list (using a TreeView) of all of the possible scopes that the variable can come from, and upon selecting a scope, shows all of the data values it can take and a short description of what it does. Internally, each scope is hard-coded as a **"ScopeDefinition"**, with its name, description, group and id. They provide a function which determines if it is a partial match with any entered strings, which drives the search functionality. Whenever the text in the search box is changed, the contents of the scope TreeView change to reflect all potential matches in real-time, making it easy for the user to see when they can stop typing early.

This dialog has been designed in such a way that you can simply type in just parts of name of the scope/data you want, and the window will immediately show and select the best match. This is especially useful for power users since this selector is used incredibly often - specifically as the driving force of the "Expression" primitive. A user can simply partial match the scope name, press enter, partially match the data, hit enter again, and they will have selected a variable in less than 5 seconds. This is much improved from Edith and Codex, which had you select variables manually from often hundred item long combo boxes.

6.6.3 Example: Animation Selector

Another specialized popup selector, the **Animation Selector** (used for the Animate Sim primitive), is special for a variety of other reasons. Immediately noticeable is the animation preview central to the window, which lets the user preview any animation without having to run it in-game or know in advance what it does. However, more important is the full search through *all animation resources in the game* available on the left, and the ability to add them to the owner object's animation set at any time. The user can select any animation in the object's animation set, which will return the corresponding ID to the operand editor as the new ID.

The central display utilises FreeSO's world renderer directly, routed through a UIExternalController which redraws every frame. The window starts and runs a temporary SimAntics VM, which only contains one sim with no thread. Animations selected on the UI are then played on this sim directly, using existing FreeSO functionality. It's elements like this that showcase the true power of UIExternalControllers, and validate the choice to utilise Monogame for specific UI elements. The only extra code which had to be written was the game UI wrapper for the world view, the UIAvatarAnimator (subclasses UIInteractiveDGRP, which will be explained later), and its interface to allow playback of arbitrary animations.

The owner object's animation set is actually an STR# chunk resource in itself, so this selector technically doubles as a resource editor. When an animation is added to the set, a string for its name is actually added to the STR# resource in question, using ResActions. The normal STR# resource editor will be shown later, which allows for more generic control over the string format. Note that this animation's source is specified by another operand field, "source", which changes the animation set this dialog accesses.

6.7 Live Debug and Step Through

6.7.1 Runtime Changes

To speed up script execution, FreeSO parses, compiles and caches BHAVs into "VMRoutine" objects. If we just simply changed a BHAV in use, calls to it would *run the cached script which has not been affected by our changes*. To alleviate this issue, we need to signal to the VM that the BHAV has changed, which will remove its VMRoutine from the cache and force a recompile the next time a SimAntics thread attempts to run the script.

All BHAVCommands which cause a functional change to a BHAV call also issue a message to the VM's static routine cache, which removes the specified BHAV from the routine cache, increments its "RuntimeVersion" field and notifies all active VMs that a BHAV has changed. Each VM notifies all of the threads it contains that the BHAV has changed, and each thread looks over its stack to see if any of its VMRoutines are out of date (its RuntimeVersion is less than its BHAV's). Whenever this happens, the thread queries the static routine cache for an updated version of the BHAV's routine, which will now reflect the most recent changes. This is not entirely optimal, since all routines in all threads have to be checked for updates even if just one changed, but the performance impact is currently negligible.

VMThreads had to be slightly modified to support breakpoints and step-through. Threads now have a **"ThreadBreakMode"**, which determines how the thread should behave when it tries to continue, steps into a function, steps out of a function, and moves to another primitive. For example, if you set its mode to "Pause", the thread will simply not advance. If you set its mode to "Step In", the thread will breakpoint after the next primitive is run. If you set its mode to "Step Out", the thread will break when it returns from a stack frame, and "Step Over" will only break for primitives in the current stack frame, or if the frame returns. If you set its mode to "Return True/False", the next time it advances the top stack frame will immediately return the specified result, and the thread will break again. The Tracer utilizes these to control execution of threads at the user's whim - and each breakpoint raised is signalled back to the IDE.

6.7.2 Tracer

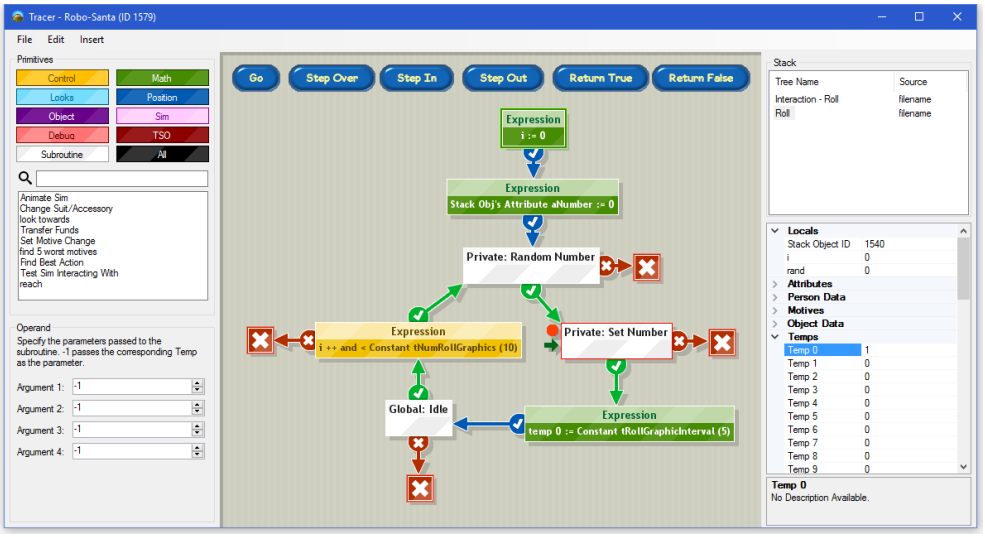


Figure 6.6: A simple BHAV in the Tracer. Temp 0 has been set to a random number, which we are able to see and change before the object's graphic is set to display it.

When a thread hits a breakpoint, a special "Tracer" variation of the BHAV Editor is opened by the IDE for it. Instead of being passed a BHAV, the BHAV Editor is given the object that threw the breakpoint, and maintains a set of UIBHAVEditors for each BHAV in its thread's stack frames. On the right of the editor some extra controls appear - a summary of the thread's stack when it was last paused, and a modifiable PropertyGrid displaying most variables that the object's behaviour has access to, categorised by source. The Tracer also introduces a number of step-through controls, which give you fine grained control over the thread's execution from any paused state.

When an object hits a breakpoint, its tracer immediately selects its top stack frame on the stack list. When an item is selected on the stack list, the BHAV tree view changes to show the code for its BHAV, and the next primitive to be executed in the frame is pointed out by a small arrow beside it. When execution is resumed or stepped through, the stack list and variable display are disabled, as their contents cannot be guaranteed while a script is running. These are updated and re-enabled when execution stops next, either due to a breakpoint, step through or a user initiated thread pause.

The step-through controls should be familiar if you have used an IDE with a debugger such as Visual Studio before. **"Step Into"** steps through primitives and into functions, **"Step Over"** does *not step into functions*, and **"Step Out"** *only* breaks when it leaves the current function. Go/Pause simply start and stop the thread on demand, and Return True/False allows the user to instantly return true or false from the current function. With these functions, it is possible to trace through BHAV execution easily.

Unlike Edith, it is still possible to add, remove, and modify existing primitives while in the Tracer window.

This lets the user make last minute changes at runtime, if they see that a certain code path is not going to work with the current input or discover a better solution for a problem. Right now, it's a little unsafe to remove primitives while a thread is running a script (the "instruction pointer" can be shifted to point to the wrong instruction), but this can be solved in the future. Volcanic is the only publicly available tool to provide this crucial functionality.

6.7.3 Live Variable Modification

As discussed above, the Tracer utilises the Windows Forms **"PropertyGrid"** control to allow the user to view and edit variables with names, descriptions and categories. The functionality of this control is intuitive to users of Visual Studio users, such as C# developers, so utilizing it for this purpose was the obvious thing to do. However - its intended purpose is the display and edit of parameters *of a single C# object*, not of remote values from many different locations!

To circumvent this issue, I used a proxy object called **"PropGridVMData"** which implements **"ICustomTypeDescriptor"**. Implementing this interface lets us modify how the PropertyGrid retrieves its "PropertyDescriptors", which define how to get/set a property's value, its name, its description and it category. For this purpose, I have implemented a subclass of PropertyDescriptor, **"VMDataPropertyDescriptor"** which provides the specified remote data as well as a custom name and description, using the contextual information provided to it. Any sets to the data by the PropertyDescriptor are made in a thread safe manner.

The PropertyGrid is given an instance of PropGridVMData as its target object, which is given the current editor scope, stack frame, and target entity. When it is queried for its properties, it automatically generates sets of VMDataPropertyDescriptors for each desired variable scope (using Editor Scope to get all of the data names), and returns them. These property descriptors are also made aware of the editor scope, stack frame and target entity, so that they know where to access the more context sensitive variables from. These artificial properties are then displayed, categorised and editable on the PropertyGrid.

This is incredibly dense, but the result is a powerful interface that didn't take much total time to implement, and contains very little boilerplate or duplicated code.

Chapter 7

Iff Resource Editor

7.1 The IffResComponent and ResourceControls

In the primer, I explained that objects heavily utilised various resources within their IFF file to aid function, such as Tree Tables, which define interactions, Strings, which can be used for dialog popups, animation references etc, and Sprites. Thus, it's important to have a facility to view, edit, add and remove these resources for a given IFF file, be it an object IFF, semi-global IFF or the global IFF itself. For this purpose, I defined a custom self-managing component called the **"IffResComponent"**.

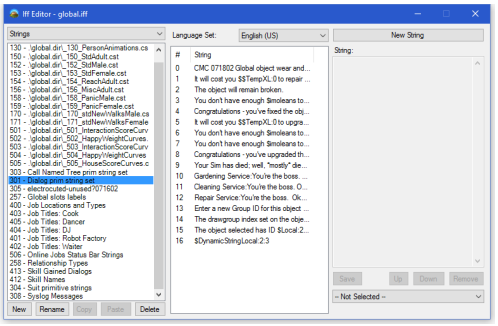


Figure 7.1: The IFFResComponent showing a string resource inside global.iff

The UI simply lists the ID and names of all resources of a specified type, and on selection provides an editor for them on the right. You can sort the list either by ID or alphabetically by right clicking on it and selecting the desired sort mode. The user can select which type of resources to list using the combo box above the list. IffResComponents also let users add entirely new chunks to the target IFF, and rename or delete existing ones. When a new chunk is created, a blank instance of the specified resource type is created using its default parameterless constructor, and added to the IFF file with the specified name and ID (assuming no conflicts!). I initially intended to allow copy and paste of entire chunks, but this functionality did not make the final version of the program.

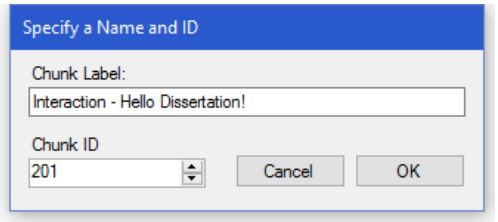


Figure 7.2: Adding a new chunk.

Editors for IFF chunk resources are embedded directly in the component on selection, allowing the user to quickly glance through the contents of many without having to open and close many windows. These editors must be told which chunk they are editing, the IFF it belongs to (in FreeSO's "GameIffResource" form, which allows generic access of groups of IFFs as one), the currently active object (if applicable) to use for context, and the chunk type's "OBJDSelectors", which will be explained shortly. All editors must implement the IResourceControl interface so that the IffResComponent can pass this information to them.

Occasionally, certain resources are flagged as important to a specific object in its Object Definition resource - for example, each object have one tree table which defines its actions, or one string resource that defines its animation table. Normally, you would have to manually enter references to these resources in the DGRP by ID. It

may also be of interest to use the IffResComponent to select a resource from an iff file, such as selecting a sprite for use in a DGRP. For this purpose, I have created **OBJDSelectors,** and the OBJDSelectorControl to represent them in the UI. The IffResComponent defines a number of "OBJDSelectors", generally linking to OBJD property names, which resources of each type can be assigned as. These are passed to the editor control, which should have an OBJDSelectorControl to let you select that resource as desired. Assuming an active object has been passed, assigning a resource to an OBJDSelector will set the property on that object's OBJD by the specified name to its ID. All resources currently pointed to by an OBJDSelector will be highlighted purple on the resource list, so they can be found easily.

The IffResComponent is entirely modular, meaning that it can be embedded in many different locations at ease. Right now, these include the Object Window, the Standalone Iff Window (as seen in **Figure 7.1**), and the SPR2 selector dialog used by the DGRP editor, which you will see later. Generally, these editors access and display the information of chunks directly, with modifications occurring via ResActions so that they are both thread safe and recorded by the change manager. The remainder of this chapter will cover the 3 currently implemented: STR#, TTAB and SPR2.

7.2 STR# Editor

String resources are used for a variety of purposes, from naming animations used by an object, to dialog pop-ups, to its name and description in the catalog (using STR# aliased as "CTSS", or Catalog Strings). They are incredibly simple - STR# chunks simply contain a number of indexed string entries, which have a string content (and optionally a string description) for a number of "Language Sets". It is only the contents of a string that affect the game's function itself, so the editor does not yet support viewing or modifying these comments.

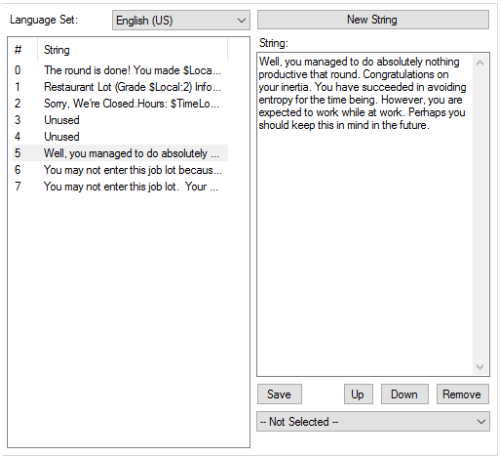


Figure 7.3: The STR# resource editor.

The editor itself simply displays all the strings in the resource, with their IDs, in a ListView to on the left of the control. When the user clicks one of the strings, it is shown in full on the right, where the user has the opportunity to modify, then save it. The user can also add new strings, and reorder/add/remove existing ones. The user can switch the active Language Set using the combobox at the top - though it is important to note that for most resources, non-english language sets are not initialized. When this occurs, the interface queries if you want to initialize it, which will fill it with a copy of the string content from English (US) as a starting point.

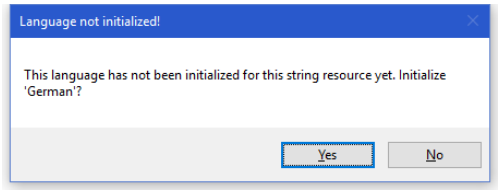


Figure 7.4: Adding a new Language Set.

All of the actions in this control, such as editing strings, add/remove or reorder, use ResActions to apply their changes on the game thread and register them with the change manager. These are sent using **BlockingResMod**, so that when a change is made, the UI *waits until it is applied* so that it can repopulate the string list with the changed state of the resource. A string can also be selected as that object's Animation Table or Body Strings via the OBJDSelectorControl on the bottom right.

7.3 TTAB Editor

As explained in the Primer, Tree Table resources define the Interactions that a given object exposes, to both the UI (via the Pie Menu interface) and the "Push Interaction" primitive. These are generally queued on the end caller object's Interaction Queue, though they can have a number of flags which can change their behaviour, or if they are "allowed" for certain callers. Each TTAB in an IFF also has a corresponding TTAs resource with the same ID, which uses the same format as STR#, but is used to provide multi-language names for each interaction.

For the Tree Table editor, I attempted to compact the layout, as the Edith and Codex variations contained a lot of wasted space, but otherwise did not change much about the functionality. I added an experimental **"Pie Menu View"** display, which displays interactions in a TreeView, using the pie menu categories derived from their names. The idea of this was to let the user see the structure of the menu they were building, but the limited vertical space seems to make it less effective than it could be.

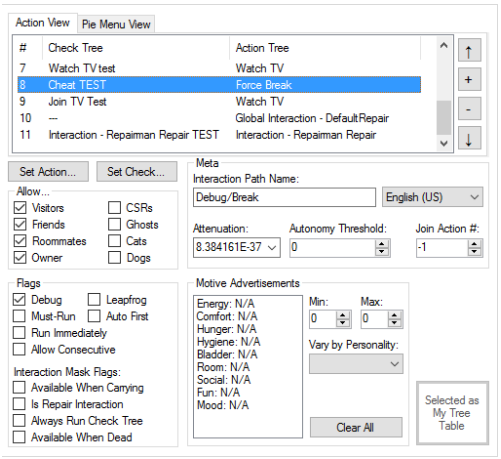


Figure 7.5: The TTAB resource editor.

When you first open the Tree Table editor, it attempts to link with the relevant TTAs resource. If it cannot be found (eg. if you just made the tree table), it is *automatically created for you*. The action and pie menu views are populated with the interactions currently available in the file, and the first one is selected. When an interaction is selected, all of the flag checkboxes, the name field and the autonomy fields are set accordingly. Any modifications to these elements apply to the TTAB instantly, using ResActions. Modifications to the interaction name, action tree and check tree will also refresh the interaction lists up top when complete (using BlockingResMod to wait until changes apply before continuing).

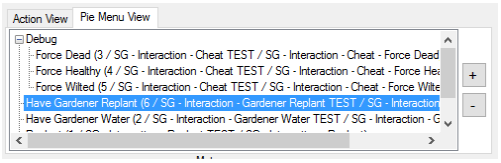


Figure 7.6: The "Pie Menu View".

You can add, remove, and reorder interactions via the buttons to the right of the list. When you add a new interaction, an entry is automatically generated for it in the TTAs resource, and the reverse applies for removals. The action and check trees can be set using the corresponding buttons, which open a simple window that lets you select a BHAV resource from the Private, Semi-Global, or Global scope to use. Selecting no BHAV lets the user clear the action/check tree. Just like the STR# resource, changing the Language Set to one that has not been initialized yet will ask the user to initialize the string set to switch to it. Finally, you can select any tree table as the current object's by using the OBJDSelectorControl on the bottom right, including TTABs in global and semiglobal resources.

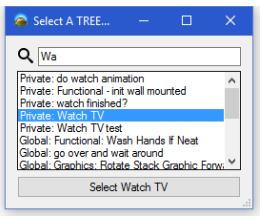


Figure 7.7: The Tree Selector.

The autonomy fields - Motive Advertisements, Attenuation and Autonomy threshold - while not important for The Sims Online, drove Free Will in The Sims. Certain interactions advertised benefits that they would provide, and sims would pick the ones best suited to their current needs. Though the display and editor here is functional, it is not important. Also, in the current version of FreeSO, most of these flags have not been reverse engineered, so they are simply dummied out, do not read correctly and have no effect. This will change over time, as more of these flags are figured out. There is also one small drawback of Tree Table Editing - you must currently delete and place down the object again if you change its tree table resource, due to some caching performed by FreeSO.

7.4 SPR2 Editor

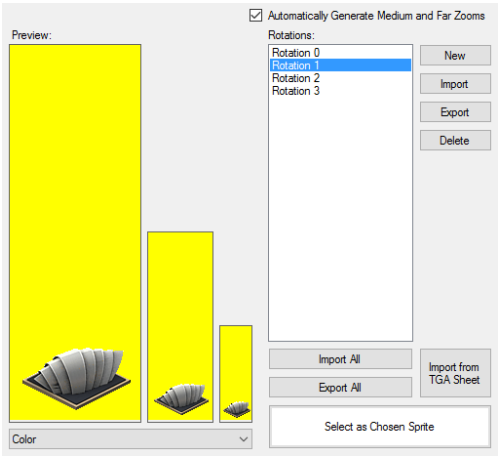


Figure 7.8: The SPR2 resource editor.

The SPR2 format is rather simple at a high level, but the underlying encoding is rather hard to understand, so it's important that the editor hides any internal complexities for the user. Each SPR2 file simply contains a number of "sprite frames", which are always a number of "rotations" repeated at 3 zoom levels. For example, a two rotation SPR2 format would contain the frames (near1, near2, med2, med2, far2, far2). It's important to consider this ordering when importing, displaying and exporting sprites.

To export a sprite rotation, its raw decoded data for each of the three channels are simply copied into .NET Bitmap objects and then saved to disk in a specified format (explained below). When copying this data out, it's important to remember that SPR2 importers actually trim the white-space around sprites, decreasing the size of the image itself from the static 136x384 (near) and supplying an offset to apply to restore its original position. When exporting, we need to generate an image with this correct static size and with the offset reapplied to the sprite data. This data can then be modified by the user, and reimported.

To import a sprite rotation, the relevant bmps are loaded in by name from a specified folder. These are read from a very specific format, split by underscores. Here is an example: "mechanicalbull_spr104_r2_far_color.bmp" The first two fields are currently ignored. The 3rd field specifies the specific rotation for that sprite, the 4th specifies its zoom distance (near, med, far), and the 5th specifies its channel (alpha, color, depth). Import *All* simply performs this for all rotations at once, ignoring field 3 and importing all rotations it can find. The transparent area of the sprite is then trimmed, and an offset is generated to place the trimmed sprite back in the original spot. However, loading the channels is only the first step - it must be encoded into the SPR2 format to be saved.

To encode an SPR2 frame, it must first be quantized to use a 255 colour palette, with one additional entry reserved for transparency. Since time constraints were a large concern, I simply included and utilized a library dubbed "SimplePaletteQuantizer" which quantizes and dithers graphics for this purpose. This palletized pixel data must then be encoded using SPR2's special run-length-encoding format which flattens all 3 channels into one line by line format. It must also point to a PALT resource, which we can simply change to our new palette if only one reference exists to it (ours). If not, we will need to generate a new one and point to that instead, removing the reference to the old palette. This reference counting for PALTs is entirely specific to them, and is dealt with on the side of the content system.

Optionally, the importer can also generate the medium and far zooms automatically, to alleviate the headache of creating them manually. You can also import from a "TGA Sprite Sheet", which is the format output by the Maxis official 3DS Max sprite exporter plugin. Overall, the tool is designed so that you can use it either like The Sims Transmogrifier, by exporting, modifying then reimporting sprites, or by importing entirely computer-generated sprites rendered using 3D modelling tools. The export format also closely matches Transmogrifier - so people familiar with it should feel at home and won't have to learn a new content pipeline.

You may notice the inclusion of an OBJDSelectorControl in **Figure 7.8** , but note that *it has not been provided a normal OBJDSelector*! This is a sprite resource as seen in a special **"SPR2SelectorDialog"**, which has hijacked the OBJDSelector interface to allow the user to select a sprite from a specified IFF resource to pass back to its parent window. To allow behaviour like this, OBJDSelectors have an optional "Callback" parameter, which is called when the selection occurs. This callback provides the selected IffChunk to any listeners, and in combination with supplying no OBJD parameter name, allows for entirely different functionality that does not touch the OBJD at all. The SPR2SelectorDialog simply overwrites the default set of OBJDSelectors for sprites (which is actually empty) with its single custom selector, which closes the window and passes back the result.

Chapter 8

Object Windows

8.1 Object Browser

The Object Browser was designed to be as simple as possible, so that users of all technical backgrounds can find the objects and resources that they want to view/edit, no matter what the purpose. The user should be able to browse and search through all the objects in the game, select what they want and immediately open an Object Window to edit its resources. Of course, it's also of interest to spawn the selected object in the VM we're connected to (as they may not appear in the in-game catalog), and it should be possible to make new objects from this window.

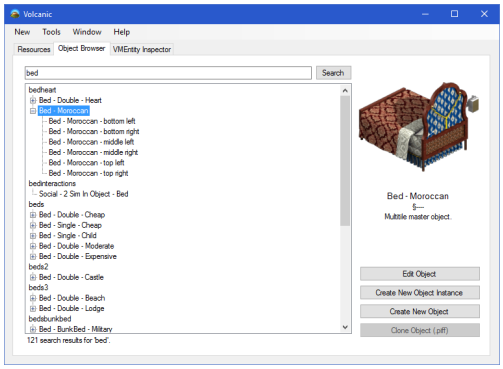


Figure 8.1: The Volcanic Object Browser.

The main focus of the display is the object list, which utilises a TreeView to group objects by file and multi-tile group. These are retrieved directly from the FreeSO content system, and dynamically update every time the window regains focus. This is because other windows in the IDE have the potential to add, remove, or change objects in such a way that changes the state of the object registry. You can search through these objects by filename, master object name, or single tile object name. Right now, you cannot search by catalog name, which left some users of the evaluation confused when they couldn't find the object they were looking for. This can be added in the future.

When an object is selected, a thumbnail of its appearance is generated and displayed to the right of the list. This is generated using facilities provided by FreeSO's LotView library, and has been wrapped within a UIExternalContainer called **"UIThumbnailRenderer"**. This control simply makes a temporary SimAntics VM, creates an instance of the preview object within it, then uses the internal function that FreeSO itself uses to generate thumbnails for objects on its UI. This is then cached and drawn out on demand. The only functionality it exposes is the ability to set the displayed object ID, everything else is handled internally. This is embedded in the UI as an **ObjThumbnailControl** - a subclass of FSOUIComponent which contains the UIExternalContainer.

When "Edit Object" is clicked, an object window is opened for object that is selected... or, if it is already open, focus is switched to it (thanks to the Window Manager!). When "Create New Object Instance" is clicked, the IDE attempts to spawn the specified object in the centre tile of the FreeSO game window. When "Create New Object" is clicked, the interface first asks you what the name of its IFF should be, then asks you what its Name and GUID should be. Upon confirmation the file is created, object is registered with the FreeSO content system, and an Object Window is opened for it.

Compared to Codex, where you cannot search or even browse objects by name, this is a breath of fresh air.

You may notice that there is a lot less functionality here than there is on Edith. Most of this related to individual objects themselves, so were moved to the Object Window instead. This leaves the object browser much cleaner, and keeps related concerns together.

8.2 Object Window

When designing Volcanic, one of the main concerns was to *reaffirm the direct connection between objects and the resources important to them*. For this purpose, all resources related to a object are grouped and editable in a specialized "Object Window". This window allows the user to view and edit all objects within a specific object IFF, contains an IffResComponent to let the user edit any other resources in it, and offers special context sensitive editors for object specific resources like DGRP, OBJf and OBJD.

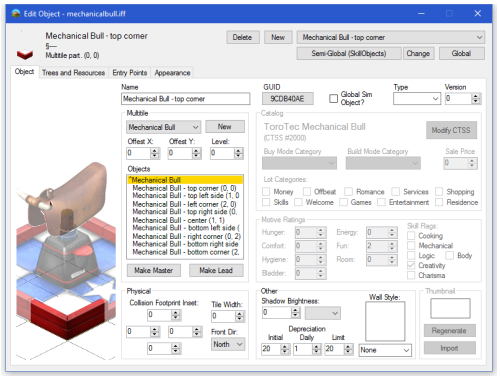


Figure 8.2: The Object Window, showing its OBJD Editor.

All resource editors within or spawned from the object window are aware of its currently active object, so that they can gain some contextual information (eg. BHAVs) or modify its OBJD with a reference to itself (OBJDSelectors). This currently active object can be switched to another in the Object IFF at any time, which notifies all editors that the active object has changed. The user can also delete and add new objects within the object IFF using the relevant buttons. Adding an object will ask the user to provide a name and GUID, just like adding from the Object Browser, but it will not ask for a name for a new IFF as the object will be added to the Object Window's IFF.

Note that when multi-tile objects are present in a file, it will contain both their master (denoted by a ^before its name) and their single tile parts (following the master, with an indent before the name) as individually selectable objects. Handling for whether or not a master is selected should be handled by the editors individually. The object window also lets the user open standalone IffResourceViewer windows for its Global and Semi-global (where applicable) resources. The semi-global resource of an object can be changed via the "change" button, which will simply ask for the name of the Semi-Global IFF file the object should use.

The rest of this chapter will cover the object specific resource editors - ones that are directly part of the Object Window itself rather than using the IffResComponent. No other editor displays resources in a manner this sensible when it comes to object IFFs - they are typically treated like any other IFF file - just collections of disconnected resources with the occasional cross reference. Volcanic's approach better fits the reality of The Sims' content system.

8.3 OBJD Editor

As a quick refresher, in the primer I mentioned that objects in the game are entirely defined by their Object Definition, which exists in some Object IFF file as an OBJD chunk. The "Object" tab allows the user to edit the self-contained parameters of that object - that is, the properties not defined by other resources. Generally, these can be split into two categories: "Physical" properties, which define how the object interacts with the world in an individual basis, and "Metadata" properties, which define how the object is registered with in the catalog and treated as a whole. For single-tile objects, both of these are relevant, however for multi-tile the concern is split between the "master" and each "multi-tile part". "master" OBJDs simply group all of the parts of a multi-tile object and provide shared "Metadata" properties for the group as a whole, while the individual multi-tile parts

provide their own “Physical” properties. For these objects, the data that it does not deal with has editing and display disabled, to prevent confusion for the user.

Changing the name, GUID or any multi-tile information of an object will cause it to be removed and then re-added to the object registry, to keep it up to date. Also shown is the catalog name for an object, which if you attempt to edit, will bring you to the relevant CTSS resource in the Trees and Resources tab. As well as showing the various OBJD properties for the active object, it is displayed on the left using a UIExternalContainer called **UIInteractiveDGRP**, which will be explained in detail later. The object display on the left will also show the entire multi-tile object any multi-tile part belongs to, but non-active parts will be semi-transparent and not centred.

From this tab, the user can also manage the Multi-Tile group membership of an object. This includes making new “master” objects to make new groups, changing the group an object is part of, and changing its tile offset in the multi-tile group (this should update the object display). This functionality is not as complete and easy to use as I would like, but it does partially function right now (there are a few bugs involving the registration of changed multi-tile objects which need to be looked over). You can also make a certain object the “Lead” object in a multi-tile group, which will set it as the object’s rotation anchor and allow BHAVs to access it via various “My Lead Object’s x” variable scopes. This functionality is rather special in that no other tool provides a way to automatically manage groups like this - Edith and Codex require you to manually edit the cross references in each object field by field to set them up as a multi-tile group.

There is not currently a way to add, remove or change named Attributes for an object, which is admittedly an oversight. The Sims lets the user dynamically create more unnamed attributes at runtime, though, so it is not a severe issue. Also, many of these fields do not yet function, such as the “Wall Style” and “Thumbnail” fields. These are a work in progress - but again, they are not necessary for an object to function correctly. At some point in the future, it will be possible to automatically generate a thumbnail for any given object.

Many of the fields in the OBJD reference the IDs of resources within the IFF, such as which tree table the object should use, or its animation table. As I mentioned in the previous chapter, these are set by the IffResComponent’s editors, using OBJDSelectors. The other tabs in the Object Window also manage resources of this kind, but do not use the IffResComponent or OBJDSelectors, as they require more specific handling.

8.4 OBJf Editor

OBJf resources define the “Entry Point” BHAVs of a specific object. They are incredibly simple, in that they only contain a list of action/check trees all entry points that an object could potentially utilise. The editor simply lists all of these entry points, and the names of any BHAVs assigned to their check or action trees. Action trees are run when their entry point is called from somewhere, either by an object or the game engine itself. Check trees, if present, run before executing an entry point to see if it is possible for the action tree to run.

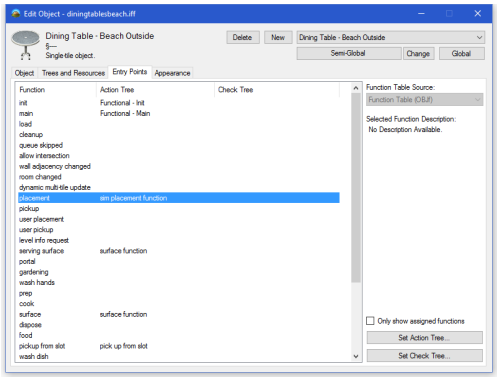


Figure 8.3: The OBJf “Entry Points” Editor.

To change the entry points for an object, the user simply has to select one, then click “Set Action/Check”, which opens up a tree selector identical to the one used by the TTAB editor. This interface has not changed much from Edith, apart from the tree selector with search, as it was too simple to demand a redesign. There is also the possibility of providing descriptions for each entry point, but it is not currently utilised.

The Init and Main functions are especially important, as they are required for every object in the game. An

Init tree should initialize the object's physical properties, appearance, and any internal attributes that it keeps track of, and is run when the object is created. The Main tree should constantly on that object's thread - allowing the object to run periodic tasks such as animation or condition sensitive changes (eg. plants die if they haven't been watered in a long time). When these are not assigned, they are highlighted red to bring the user's attention to them.

Entry points can also be sourced from the Object Definition, if its "Use OBJf" field is set to 0. To support both of these sources, I set up two simple functions to get the function table for an object for display, and one to set an entry in it. Depending on this flag, these functions switch to access the correct resource. Note that when the OBJD defines the entry points used, they cannot have any check trees, so the button to set an entry point's check tree is disabled. It was intended for the user to be able to automatically be able to generate an OBJf from an OBJD, but it could not be included in the final version.

8.5 DGRP Editor

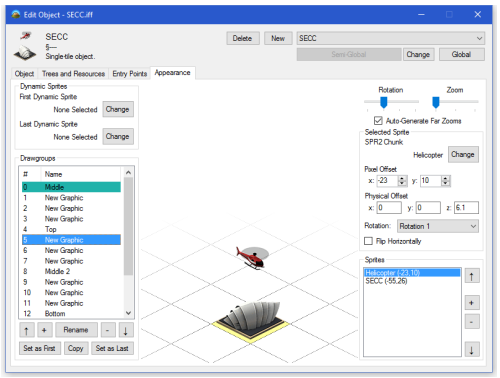


Figure 8.4: The Drawgroup Editor.

Drawgroups in The Sims contain arrangements of sprites for each zoom and rotation, and define unique graphical states that an object can utilise. Each object in the game must specify a "Base Graphic" - the ID of a DGRP chunk to be used as its 0th graphic, and a number of graphics to define the range of DGRPs that its BHAVs can access. These graphics can be switched between or cycled by BHAV scripts, either for animation purposes or to display a new state.

Each Drawgroup contains a **"DGRPImage"** for each zoom and rotation in the game (3 zooms, 4 rotations). Each DGRPImage contains a number of sprites, drawn in order, which together form the complete image which is displayed ingame. These sprites have a number of parameters: obviously the SPR2 resource to use as their graphic, the rotation of it to use, if it is flipped, a screen offset, and a "physical offset", which moves the sprite in the specified relative direction in the world. It is important that when the user edits these, they can see the results immediately in a preview window so that they can fine tune graphic positions to match up perfectly.

In the centre of the view is a UIExternalController driven object view, the **InteractiveDGRPControl** using the same kind of configuration as the Animation Selector shown in the BHAV Editor section. The view actually places the active object in a temporary SimAntics VM, which is rendered using the FreeSO LotView libraries. The **UIInteractiveDGRP** element, which manages this temporary VM and its rendering, exposes a number of functions to let the IDE manage its display:

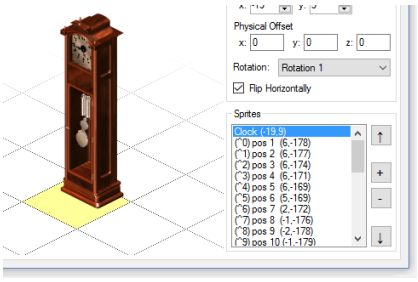
```
public void SetGUID(uint id) //changes the active object being displayed
public void ChangeWorld(int rotation, int zoom)
public void ChangeGraphic(int gfx) //changes the DGRP being displayed
public void ForceUpdate()
public void SetDynamic(int i) //sets the active dynamic sprite
```

These functions are reflected in a thread safe way by InteractiveDGRPControl for the Windows Forms interface to use. SetGUID is called first, which should pass in the current active object to initialize the display. ChangeWorld is called when the rotation and zoom sliders are changed, ChangeGraphic is called when the active DGRP is changed, and ForceUpdate is called when any element of the DGRP is changed. SetDynamic is called when a "dynamic" sprite is selected on the sprite list, forcing it to show up on the display. This will be explained later.

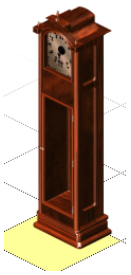
On the left of the display is a list of all of the DGRP resources in the object IFF, which shows both their name and their graphic index for the current object (ID - Base ID). The base graphic for the active object is highlighted green, the end graphic is highlighted red, and any outwith the range are grayed out but still accessible. The user can easily add, remove, reorder and rename DGRP resources in the object using the controls below the list. It's also possible to set the active object's base and end graphics to any DGRP, which will update the IDs and any highlights the list is displaying accordingly. During development, I found that it was common for a user to wish to duplicate an existing graphic to perform animation using the physical offset, so I also made this possible via the "copy" button.

When a DGRP is selected, the sprite within for the current image (using current rotation+zoom) are shown in the sprites list, showing their SPR2 name and pixel offset. When you click on one of these, you can see its full set of properties in the "selected sprite" box, and simply modify them using the controls provided. When any change is made, it is reflected immediately ingame and by the central preview. When you wish to change the target SPR2, an **SPR2SelectorDialog** is displayed, which utilises the IffResComponent to let you add, remove and modify sprites as well as select one via a custom OBJDSelector. (you may remember this from the previous chapter!)

Since the order of sprites is important for transparency, I have made it easy to reorder sprites within the image, using the up and down buttons next to the sprite list. To reduce the workload of editing all 12 DGRP images for a graphic, I have made it possible for the interface to automatically generate the medium and far zooms (with adjusted pixel offsets), leaving only 4 images for the user to populate with sprites. This has not been done for rotations, as a common use case is for sprites to only have two rotations and flip between rotations, but it may be possible in future to "guess" what the user wants depending on how many rotations are available in the SPR2.



(a) Clock with Dynamic Sprites



(b) Dyn sprite for :30

(c) Dyn sprite for :40

(d) Dyn sprite for :50

It is also possible to perform some rudimentary modification of **"dynamic sprites"** for an object; sprites which only appear if their corresponding **"dynamic sprite flag"** in the object state is set. For example, the "Grandfather Clock" object's graphic contains 12 dynamic sprites for both the large and small clock hands, which the object script shows/hides periodically to show the current in-game time by toggling the relevant sprite flags. The range of SPR2 resources is set in the OBJD for that object, again using a base ID and total number, which the interface lets you set using the SPR2SelectorDialog. In the Sprites view for a DGRPImage, sprites which are dynamic will show beside them their flag - eg. (^1) - and selecting them on the list will force their dynamic sprite flag to on and all others to off for the central display. This is rather simple for the time being, and should ideally be extended to allow toggling of multiple dynamic sprite flags for the display.

The Drawgroup editor is unique in that it is the only existing tool of its kind and scope. The interface was designed from scratch, without existing reference, to fit the format as best as it could. The result is a streamlined visual editor that allows the user to modify the entirety of a Drawgroup at a whim, and see their changes apply in real-time within the live preview. The closest competitor, Transmogrifier, requires the user to manually modify and import Drawgroups externally via an XML format, or most likely, just leave the user to stick with the ones provided by the object they cloned. For this reason, there are currently very few custom The Sims objects which utilise specialised Drawgroups. Volcanic not only makes it possible, but simple to make entirely new Drawgroups from scratch, and this is definitely one of it's strongest points.

Chapter 9

User Evaluation

9.1 Approach

While developing Volcanic, I decided that to truly determine whether or not I had succeeded at meeting the aims of the project, I would need to run a User Evaluation. This evaluation would need to test uninitiated users' ability to modify and create The Sims Online objects using the tool, and ideally highlight the main issues with its current state. Of course, we also want to know what people liked about the tool, especially compared to the currently available alternatives.

You might be initially sceptical that an evaluation for a project like this would get any participants at all, since the target platform "The Sims Online" shut down 8 years ago... but since then, the re-implementation project FreeSO has garnered a sizeable community of dedicated The Sims Online fans. The most active members of this community are both very technically inclined (considering the game engine is rather difficult to use right now), and long term fans of The Sims and The Sims Online, with a deep understanding of their gameplay and the concepts behind it. This makes them the perfect candidates for a User Evaluation, assuming that there is enough incentive to participate.

Because of the number of different things you can do in Volcanic, it was not sufficient to provide the program to users and simply tell them to "have at it". To cover as many bases as possible, I decided to split the evaluation into 5 unique tasks, each demonstrating one action that a typical user of Volcanic might like to perform. I decided to make each of these tasks optional, since some are more difficult than others, but offer a 3 tiered in-game reward object to encourage users to participate in as many tasks as they could. For each task a participant would provide a summary of what they achieved, a list of things they liked and any problems they ran into. Feedback for each task is therefore specific instead of general, and better identifies the unique problems with performing different actions.

Finally, after completing all the tasks that they can, the user would fill in a survey regarding all areas of Volcanic. This survey was intended to cover a number of things that the tasks alone could not - determining the characteristics of the evaluation participants (programming, SimAntics background), their motivations, how they feel about Volcanic in direct comparison to the other tools, and what planned future features they value the most. The survey also contained some more specific questions about certain design decisions, perceived problems, and optional written responses for the participant to voice opinions not covered by the questions.

9.2 Tasks

For each task, a small example walk-through was given to help the user get the hang of the system. As part of the requirements for each task, the user's actions *had to be reasonably different from the example*, to demonstrate that they truly understood the task. When users completed a task, they were instructed to post on its relevant thread on the FreeSO forums. This post included a summary of how they achieved the task, a link to the resultant PIFF or IFF, and a bullet point list of what problems they faced and what they liked.

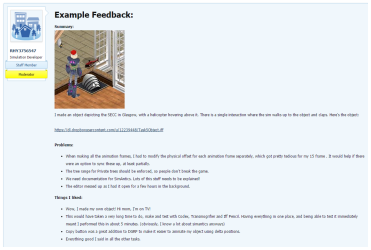
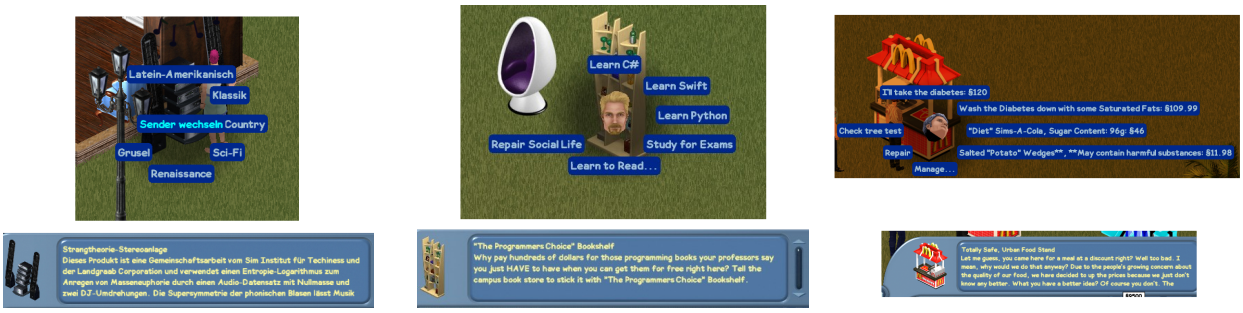


Figure 9.1: An example of the format feedback should be in, provided by myself.

Since the tool was still in development when the evaluation was launched, tasks 2 and 5 were not initially available, as the features required to complete them were not complete. The evaluation continued one week after task 5's launch, to allow most users to attempt it, but a reason for the lower participation in these tasks could simply be that users did not have the free time to complete the tasks within the shorter duration they were available.

Translate Object Strings [9/9 complete]



(a) Stereo translated for the German language.

(b) Bookcase changed to let the user "learn to program".

(c) McDonalds object with more humorous strings.

Figure 9.2: Various object strings modified by evaluation participants.

I decided that the first task should not involve any SimAntics at all, so that it could serve as an introduction to the tool and how everything is laid out. All a user had to do was open the game with a different language code (-lang launch parameter), find an object to "translate", and modify both the strings in its Tree Table and Catalog String (CTSS) resource files. The user did not have to translate the object into another language, only modify the string in such a way that demonstrates that they could. A small generic tutorial was provided for this purpose, which introduces the Object Browser, Object Window and the IffResComponent, and teaches the user about Language Sets.

Participants found it very easy to translate strings, as expected, and even commented that they appreciated the overall design of Volcanic and found the object browser very easy to use. However, there was a small problem where users were not initially aware that they were meant to click the "save" button to save their changes on the STR# editor, since changes everywhere else apply instantly. They suggested some sort of confirmation dialog on attempting to switch out of an unsaved string, though it may be a better idea to automatically save changes like the other tools do, as they can be reverted by the change manager anyways.

One user found it rather difficult to find the object they were looking for as they were attempting to search for its catalog name ("Rampa Lopside End Table"), not its internal object name ("Table - End - Vegas 1"). They also remarked that it would be nice to streamline the editor for people who only want to do translations (optionally hiding the other tools), which may be of interest to get less technically inclined folks to help with the translation efforts.

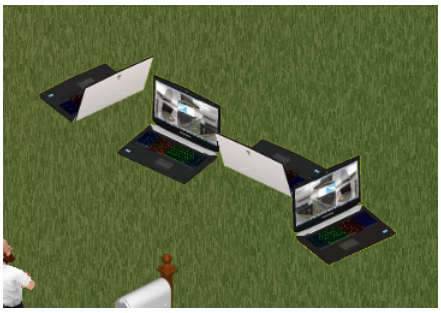
Surprisingly, even before this task was released, users already began full scale object translations into German, Brazilian Portuguese and Spanish, almost unprompted! This use case is much bigger than anticipated, and

will need a lot more focus and support in the future.

Modify Existing Object Appearance [8/9 complete]

(a) Custom TV channel by an evaluation participant.

(b) Custom DGRP with an additional sprite.



(c) "Laptop" object modified to be of the "Alienware" brand.

Figure 9.3: Various object graphics modified by evaluation participants.

This task was designed to test the user's ability to simply recolour or reskin an existing object, using both the SPR2 importer and DGRP editor. It assumes the user is able to find an object and knows the basics of the Object Window from the previous task.

People enjoyed that it was still as easy as Transmogrifier to export and import object sprites, and nobody had any real problems with the process. People also liked that the tool automatically generated the medium and far zooms for them, which reduced their workload considerably. The biggest complaints were that the separated alpha-colour channel system was a bit antiquated (could simply use alpha png nowadays), and that making sprite z-buffers as a whole was a little difficult since no 3d export tools were available yet. This task also revealed some bugs with my sprite importing - there were some cases where changes to a palette would not be correctly registered and fail to save, due to an unusual use of ResActions, causing the modified sprite to use the wrong colours entirely. These issues have been fixed.

One user even replaced the television channels on one of the TVs with entirely custom ones, simply by modifying the channel sprites the object used. Though I listed use of the DGRP editor as optional, many users gave it a shot anyways, generally adding simple sprites to float above their object, like the happy face graphic shown in **Figure 9.3b** . They did not seem to find any issues with the DGRP editor, apart from needing to specify each rotation individually.

One participant did not complete this task, not because of how difficult it was, but because it launched around about halfway into the evaluation. Thankfully, this did not affect most of the participants.

Modify Existing Object Function [9/9 complete]

The next logical step was to modify the function of an existing game object. This task involved direct modification and addition of BHAV scripts within existing objects, as well as encouraging the user to add and modify interactions within the objects' TTAB resources. No tutorial was given for this task, since I figured it might be more interesting to see what people could come up with on their own, though I did give a simple example of an object I had modified and example feedback.

Figure 9.4: "Flamingo" object modified to be clonable.

Despite the much increased technical demand, I was surprised to find that *all participants in the evaluation managed to complete this task*, with most of them making very satisfying modifications! Users generally praised the design of the BHAV editor, the coloured categorisation of primitives, and the small description

of each one's function on its operand editor. People also gave much praise to the specialised Animation Scope selector with preview, stating that it made finding an existing animation similar to what they wanted easy. Functionality like this was requested for sounds and accessories, which will definitely be a focus in the future.

There were a few issues, obviously. There is an obvious outcry for in-depth documentation covering the SimAntics environment, such as the chapter provided in this dissertation, as this was not available to the users at the time. Thankfully, *users were able to learn by example from the existing game objects*, since they could all be viewed in the tree editor. This is a very important observation, as it shows that the tool has successfully achieved what it was aiming to provide in the first place! Users were also not sure of the Chunk IDs that BHAVs were expected to use, which should have been within the range 4096-8191 for BHAVs in the Private Scope. This has since been made more obvious by the chunk addition window, which will now automatically set its accepted Chunk ID range for BHAVs to that of the detected scope (Private, Semi-Global, Global).

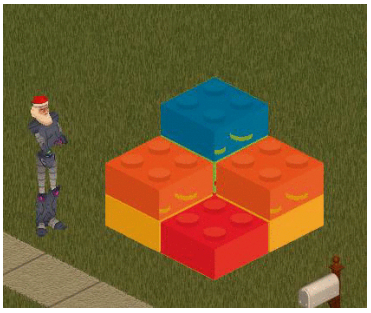
Not many of the participants used the debug features, unfortunately, but those that did found that they were powerful enough for their needs, with a few small glitches in the display and functionality. These have since been fixed. There were also a few feature suggestions, such as copy and paste of primitives (between trees too!), an option to delete all primitives in a BHAV, duplicating entire BHAV chunks for modification, and obviously support for the primitives currently missing operand editors. Most importantly, people were sort of surprised at how easy it actually was to modify an existing object, which inspired them to complete Task 5 later.

Import TS1 content into FreeSO and fix any broken BHAVs [2/9 complete]

This task was meant to test a deeper understanding of SimAntics, but ultimately failed miserably. The idea behind it was that several TS1 objects "exist" in The Sims Online, but do not work correctly as some of their cross references are invalid. The task was to find one of these objects, look through the BHAVs to find the areas that contained invalid references/function calls. The idea was that this would require use of the Tracer to debug troublesome scripts and for the user to respond to any SimAntics exceptions to occur, but this turned out to be too complex for most users to understand. Another problem with this is that it was not entirely obvious which of these objects could even be fixed easily, or what the specific problem with each was. Only 2 people managed to complete the task, apart from myself.

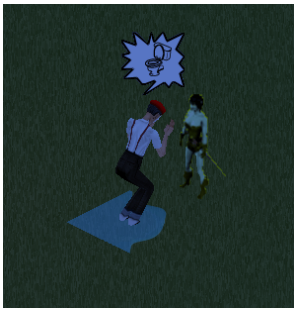
Because of this, I modified the requirements for the reward so that the user would only need to complete task 5 to achieve the "gold" level event object reward, as it covers the sections of the editor the user would need to touch here anyways.

Create an Object entirely from scratch [7/9 complete]



(a) "Lego Block" object, made by a participant and modified by me to be stackable.

(b) "Fortune Teller" object, which gives the sim a desired object.



(c) "Potion Seller" object, which causes sim to perform a random action after drinking a potion.

Figure 9.5: Objects completed by various evaluation participants for the fifth task.

For the final task, the participants were asked to create an object entirely from scratch, creating and providing

all of the required resources themselves. This was the task I expected people to perform last, as it is essentially all of the other tasks put together, with some intermediary steps to connect their new resources to the object. Because I saw this as the most difficult of the 5 tasks (and the participants had little prior SimAntics experience), I expected less than half of the participants to attempt it, however I was very pleasantly surprised when 7 out of 9 managed to complete it!

I provided a simple walk-through of the process that a user would need to do to create a reliable object from scratch. This starts with creating the IFF from the Object Browser, then creation of the object graphics using the SPR2 and DGRP editors (which using is no longer optional!), then giving the tree the required "init" and "main" entry points, then finally setting up an interaction for it as before. The whole process was intended to take around 10 minutes to complete, though obviously this could not be timed for each participant.

The greatest praise came from the users that did not expect the process to be as easy as it was. They did not expect to be able to complete the task, generally since previous experience with other tools left them confused, but were pleasantly surprised when they found it was possible to complete entirely within Volcanic by employing a simple process. More participants praised the ability to edit and test BHAVs immediately within the game, saving them a lot of time and not requiring any saves to changes still in progress.

There were a few more detailed criticisms resultant from this task, which makes sense given its increased scope. Again, users were longing for some kind of in-depth documentation to cover the basics and general practices of SimAntics, which was not yet available. People found that many of the editors being in different places was a little confusing, but found that it became easier as they used the tool and understood the environment. One user was not able to easily determine which rotation the "front" of an object was on, and suggested a visual ground arrow to show this in the DGRP editor. Another requested some way to automatically generate the 4 rotation images for a DGRP based on the SPR2 rotations present, which was something I expected to see, and am definitely going to look into.

One user managed to create a "lego block" object that you could recolour and duplicate using interactions. Upon duplication, the object would appear in the sim's hand. What's surprising is that this user found out the standard procedure for getting a sim to carry an object without immediately placing it down, simply *by looking at Maxis objects for example*. This user then contacted me directly to ask about the possibility of making the lego blocks stackable, but I replied that this required a resource editor for the "SLOT" type chunk - which defines the positions at which an object can contain another (eg. tables and counters). Though this editor is not available, I took their object and added this functionality with a SLOT resource I made manually with a Hex editor, which achieves the intended functionality. This "learn by example" philosophy was exactly what I was hoping for to be possible with the tool, and maybe with some more extensive documentation, everyone would be able to make powerful and impressive objects with no prior knowledge whatsoever.

Participation of this task was much greater than expected, and even more surprising when you note that the survey recorded none of the 9 participants being notably familiar with SimAntics. This could partially be due to the walk-through I provided... though judging from the vastly different result objects I believe that it was mostly due to the participants understanding how to use the tool, learning from the example I provided and their pre-existing understanding of The Sims gameplay.

9.3 Survey

The tasks were great for receiving feedback on each process individually, however I also wanted some more general feedback on Volcanic as a whole, and some investigation into the background of the users who did participate. I was mainly interested in comparing the tools directly with Codex, The Sims Transmogrifier and Edith, feature by feature to see where I've went right or wrong with the revised design.

The survey starts by asking a few questions about the participant themselves - specifically which "The Sims" related tools they have used before, their prior experience with programming/SimAntics and what their motivations are for completing the evaluation. The next 6 sections cover specific areas of Volcanic: the Object Browser, Object Window, String Editor, Tree Table Editor, BHAV Editor and DGRP Editor.

Each of these sections starts with a picture of the process in Volcanic, along with direct comparison pictures of the same process in other popular tools. The participant is then asked which tool they believe looks easier to use, which looks most powerful, and which looks like it might be most effective for "power users", or more familiarised users. Following this, the participant is asked to rank the usability of features which differentiate Volcanic from the alternative tools, from Much Worse → Worse → No Effect → Better → Much Better. The user is then asked to rank certain future features I had in mind by how important they believe their development is, from Waste of time → Neutral → Neutral → Important → Crucial. Some optional free-form text entry fields are available at the end to let the user express any specific future features they might like to see, problems with the current design, and things that they liked.

The final section, "Overview", asks the user whether they believe the development of the tool was worth it, from 1-10. It then asks the user how much documentation they believe certain portions of the tool require. The "future features" questions are also present here for some more general features (eg. SLOT editor) and suggestions. Finally, the user has a chance to voice any general opinions that they have, which admittedly most users just used to express their thanks.

9.4 Survey Results

For the full summarised results of the survey, see Appendix E. Many of the results are rather interesting, so I'll briefly summarize the most important ones here.

Firstly, the results reveal that the 9 participants of the survey actually have mixed programming backgrounds - though despite this, *only one* claims to have some pre-existing knowledge of SimAntics! The most popular reason for completing the evaluation is "for fun" at 8 ticks, with "interested in gameplay behind the scenes" and "for the event object" following close behind at 7 ticks. The "most important" reason for most was "interested in gameplay behind the scenes", however, with only 2 out of 9 citing their most important reason being their desire for the event object.

When asked about the **Object Browser**, most users agreed that Volcanic was easier and more effective to use than Edith and Codex, though there was an equal split when it came to which tool was "more powerful". For Edith, this could be attributed to the options on the browser instead of the object window, but for Codex it might just be that it "looked" more complicated. Most users claimed that they found it easy to find the objects they were looking for, with an average of 8/10 (one user voted 4, could be attributed to lack of catalog name search). Users mostly agreed that the features pointed out improved the usability of Volcanic, with the Object Thumbnail preview not present in Edith and Codex being a clear favourite. However, one user said its appearance made the experience worse, with one comment saying that Hi-DPI or zoomed screens cause the layout to malfunction. For future features, people seemed to agree that it needed the ability to search by catalog name, but for one of my requirements that I failed to meet, "Search by primitives used", users were mostly neutral. Some users failed to see the thumbnail in the Object Browser, which will need to be looked into. People seemed to fill the suggestions with content relating to other sections... perhaps the survey's structure was not explained well enough.

For the **Object Window**, most users agreed that Volcanic was easier to use, the most powerful (unanimous!) and the most effective. When asked if the OBJD editor on the main tab was too complex, I received a mixed response averaging 5/10 agreement, suggesting that it's worth looking into simplifying. People seemed to agree that the tab layout in the window was sufficient, though. Two users suggested moving "Appearance", the DGRP

editor, to be the second tab from the left, and one suggested giving sprites (SPR2) their own tab. When quizzed about some of Volcanic's unique features, users agreed that the Thumbnail Preview and embedded resource editors made the tool more usable, but *were torn on whether or not the "all in one place" solution really helped usability*. It's likely worth looking into cleaning up the layout to gain the functional benefits of an all-in-one set-up while minimizing user confusion. Users clearly desired a way to copy/paste existing resources. People were most confused about which Chunk IDs to use when creating resources, which could likely be semi-automated for most chunk types in future.

Most people actually thought the **String Editor** of Codex was easier to use then Volcanic's, though users seemed to agree that Volcanic's was more powerful and effective in the long term (despite missing the string comments feature). Users generally agreed that the features pointed out were improvements, but there were no clear favourites. However, when quizzed about potential future features, with there being a very clear demand (8/9 users voted "crucial") for "Auto-save or prompt on no save", which makes sense given the issues raised in Task 1, with "Remember language set used last" following closely. People agreed that editing strings was never really hard at all in the first place, but did again identify that the manual saving became a problem for them.

The **Tree Table Editor** was more of a point of contention for users. Though 5/9 always agreed that Volcanic was easier to use, more powerful and more effective than Codex and Edith, it was never a clear winner. People seemed to think the new features were at least a bit better than before, but there was contention on whether or not the smaller layout added or detracted to the tools usability. People seemed to agree that a better utilisation of space was something that needs to be looked into, with some users commenting that Volcanic's layout felt "cluttered". People liked that it was rather easy to change interaction names, and add categories using the separator "/" (which is actually a game feature!), but got confused which Chunk ID they were supposed to use and wished for a way to mass modify interaction names across resources. This part of the Volcanic is definitely worth looking into improving, given that it garnered the worst response of all of the components mentioned in the Survey.

For the **BHAV Editor**, most participants believed Volcanic's version was easier, more effective and more powerful than Edith and Codex. Users claimed a mix use of the "Tracer" functionality, which could be attributed to the lack of any tasks demanding its use. Most users claimed that the features highlighted greatly improved the usability of the editor, with very high praise going to the primitive group colouring, visual operand editor in the same window instead of a dialog, and the overall appearance. From a number of suggested future features, users highlighted copy/paste and a sound selector. Some features, like zoom in/out, labels/goto and BHAV comments received a lot less interest than expected, so might be held back for future development. I also threw in "A Text Based language for SimAntics" to test user interest for it - but the reception was the most mixed out of all of the features, with one saying it was a waste of time. The free-form feedback was mostly the same as in the tasks - users liked the design, found it easier to comprehend and felt that the BHAV editor was the strongest part of the program. One user claimed that "big trees could get cluttered", which could be partially solved by adding labels and a zoom feature in the future.

Most users agreed that Volcanic's **Graphics Editors** were more powerful and effective than Transmogrifier, though there was some more contest when it came to ease of use. When quizzed whether the presence of the powerful DGRP editor detracted from the simplicity of changing sprites, I was met with a rather mixed response - with 4 out of 9 voting no and 5 voting yes. From this I'm guessing it's not too much of a problem, but it's definitely worth looking into keeping the process as simple as possible for users who only want to recolour sprites. No participant said that any features highlighted made the experience worse, and most agreed that the live preview made the process much better than without. When quizzed about future features, many asked for things like "Automatic drawgroups" or "auto-generating alpha/depth channels", similar to what Transmogrifier provides. It may be possible to make a "wizard" to make these possible in a reasonable fashion in future. Participants generally praised the automatic far-zoom handling and the live preview, and found that the process for recolouring could be a little tedious.

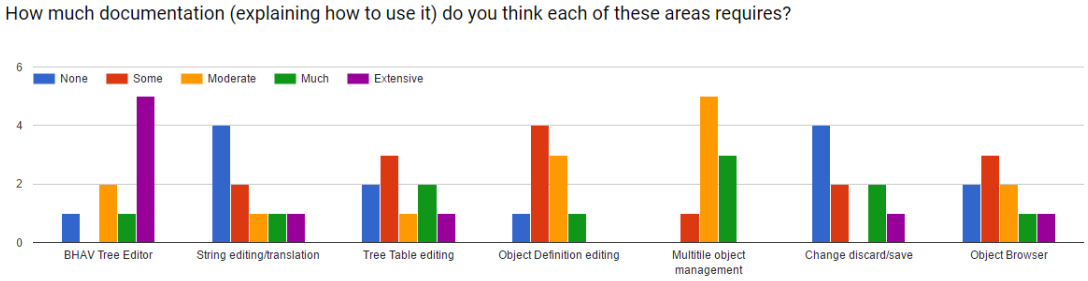


Figure 9.6: Participants ranking how much documentation they believe is necessary for various components.

Finally, most (6/9) participants agreed that Volcanic's development was completely worth it at 10/10, with 2 voting 9 and 1 voting 8. Calls for documentation were as expected across the different areas, with a clear desire for "Extensive" documentation for the BHAV Editor and "Moderate" for the management of multi-tile objects. For the string editor and the change manager, most participants claimed that they needed no documentation at all, while for others users appeared to desire at least some or had mixed opinions (specifically with the TTAB editor). As expected, most users valued documentation and tutorials the most when it came to future features, though it was surprising to find that people found "running without the game open" unimportant. A port to The Sims 1 was also a popular choice, which will definitely come in future when FreeSO itself is ported to support The Sims 1's content system.

6 of the participants left responses for the final open question, which they all used to give thanks and show their appreciation. Overall, it was pretty flattering to see the participants of the evaluation put as much work and thought into it as they did, and I would like to thank them once more for sticking with me to help make this tool the best it can be into the future.

9.5 Summary

Overall, I think the evaluation was effective in finding both the strong and weak points of Volcanic. Most importantly, it has highlighted that it has succeeded in its aim of being much easier to use and provide much more freedom when it comes to creating and editing objects. The results of the evaluation have definitely helped decide which direction the editors should be taken on in future, and have proven that there is definitely a desire for a tool like Volcanic to exist. The continued development of full object translations into Brazilian Portuguese, German and Spanish are living proof of this fact.

There were a few small problems with the evaluation as a whole. Some time after publicising the tasks list, I realised that none of them really encouraged or required use of the "Tracer" functionality. Also, users were never asked to utilise Semi-Global or Global resources, although admittedly understanding these may have been more of a SimAntics comprehension task rather than one testing a user's ability to use Volcanic.

Unfortunately, the user evaluation started too late for it to substantially influence the final design, mainly because many of the features required for a meaningful user evaluation were only usable late into Volcanic's development. This was so much of a problem that two critical features - modification of graphics and creation of new objects - had to be completed *during the evaluation*! If I were to do a similar project in the future, I would definitely plan a user evaluation closer to the middle of development, so that any problems would be identified sooner rather than later and corrected accordingly.

Chapter 10

Conclusion

10.1 Future Work

Volcanic has easily met all of the aims I outlined at the start of this report, as indicated by the evaluation. Despite this, it is not "complete" in any sense, and will continue to be developed alongside FreeSO. With further development, it should be possible for people to make truly astounding content for The Sims Online, and easily share it with the world.

It is clear that extensive documentation for the SimAntics Environment needs to exist for the tool to be truly successful, and perhaps a "General Practices" guide to inform users of how Maxis achieve commonly desired functionality in their official BHAVs. Perhaps releasing the "Primer for the SimAntics Environment" from this dissertation might be a good starting point, though a Wiki may be the best way to go about this.

Some areas of the interface definitely need cleaned up, as revealed by the evaluation. Specifically, the Tree Table editor needs a full restructuring, and the OBJD Editor needs to be simplified. Right now, the latter is more a collection of disconnected number entry fields than a coherent component. Some interface needs to be introduced to define acceptable ID ranges for each chunk type, depending on the IFF file they are created in. (Private, Semi-Global, Global)

As well as the features suggested and highlighted in the evaluation, I would also like to focus on achieving some of the requirements that I failed to complete, such as an ANIM editor, "Clone" PIFFs, a Mesh Editor and a SLOT resource editor. Though these are not important to create any new objects, they *are* essential for creating truly unique ones. I would also like to get Volcanic running on Linux and Mac, so that users don't have to switch to Windows to modify their objects when the game itself already runs on their OS.

It's definitely of interest to port the tool for use in The Sims 1, though this will require first porting the FreeSO engine to utilise The Sims 1's content. This will likely happen in the next few years - porting Volcanic to use the alternative content system may prove challenging, though. It was also initially part of the plan to get Volcanic running without the game itself installed, though this proved too challenging to achieve within the project time-frame. Judging from the response in the evaluation, this feature may not even be worth looking into, as making game objects without the game itself ultimately feels counter-intuitive.

You may notice that throughout this report, I have been deliberately avoiding explaining how sound effects work in The Sims, how they are created and how they are played. This is because they are actually driven by *their own virtual machine*, called the **"HIT VM"**, which executes a byte-code language known as HIT to sequence and play specially configured "Tracks". It should be possible to create sound effects in this language in the future, perhaps even with a Wizard to abstract away the complexities of the byte-code language.

10.2 Summary and Opinions

Drafting the requirements of the project, I never anticipated that the final product would be so substantial. Somewhere between working on it and FreeSO in my spare time, I gained a rather unhealthy drive to push the IDE development as far as it could go. I'm entirely convinced that doing so was completely worth it, both for the tool that was created as a result and for the invaluable experience that designing and developing it provided. If I did this as a hobby project, I likely wouldn't have been as driven to create an effective and future-proof design, or follow a formal software development process. If I were to do this again, I'd likely start with extending the FreeSO content system for modification first, as doing so later led to a few bugs and minor inconsistencies concerning how changes are saved throughout the code.

When I started work on this project, I had *no prior experience* of working with Windows Forms whatsoever. Now, I feel confident that I can create usable and powerful interfaces with it at ease, thanks to the in-depth interaction with its internals that making many of the tools in Volcanic demanded. Modifying an already complex content system to also support saving out, reverting changes, runtime patches and live modification proved an unparalleled challenge which really got me thinking. As a result, I feel much more confident as a C# programmer and code architect that I ever have before.

Thank you for reading this dissertation the whole way though. It's been quite a journey for me personally, both developing the tool and writing about it in such an extensive fashion. I hope you enjoyed seeing Volcanic's development process from start to finish, and I hope you give the tool itself a try!

Appendices

Appendix A

Running FreeSO and Volcanic

Currently, FreeSO and Volcanic only run reliably on Windows XP or greater. Linux and Mac have partial support, but I would not recommend using them right now.

A.1 Installing The Sims Online on Windows

To run FreeSO, you must first install a copy of The Sims Online. As of the 24th March, 2016, you can download a working installer for The Sims Online from EA's own servers here:

http://largedownloads.ea.com/pub/misc/tso/Setup%20The%20Sims%20Online.exe

Install the game on any drive, but remember the directory you install into. Proceed all the way into the full-screen installer. It will ask for a CD-Key, but the installer should progress as shown by the bar at the bottom. When the game fully installs, simply quit the installer.

Next, you need to extract some extra files into your The Sims Online directory. Extract the extra packing slips from here:

https://dl.dropboxusercontent.com/u/12239448/packingslips.rar

...into the folder you installed The Sims Online in, making sure that you overwrite the existing packingslips folder.

Finally, you must install OpenAL. Download and run the windows installer from here:

https://www.openal.org/downloads/

A.2 Running and using FreeSO/Volcanic

You can now either build the project as described below or download and run a build from the FreeSO buildbot (since Volcanic is present in the official FreeSO repository):

http://servo.freeso.org/viewLog.html?buildTypeId=ProjectDollhouse_TsoClient&buildId=lastSuccessful&buildBranch=%3Cdefault%3E&guest=1

If you're officially marking the project, you will likely want to build the version of FreeSO in the code directory, in which case you should not use the latest build of FreeSO and Volcanic linked above.

FreeSO.exe will simply start FreeSO, and FSO.IDE.exe will start FreeSO with the IDE injected into it. You should see the loading screen either way, which will transition into the create-a-sim screen.

Create any sim you wish, click accept, and you will get to the map view. From this view, you can select from a number of preset lots, most of which are job lots. Double click "empty_lot.xml" to get to an empty lot. If you have used FSO.IDE.exe, Volcanic should start alongside the lot.

A.3 Troubleshooting

If you encounter any problems, or the TSO files are no longer available from the EA site, *contact me immediately.* Make sure your graphics drivers are up to date, and you have the .NET Framework 4.0.

Appendix B

Building FreeSO and Volcanic

To build FreeSO and Volcanic, you will need Visual Studio 2015.

If you wish to build the latest version, clone the git repo at https://github.com/RHY3756547/FreeSO.git. Make sure you have initialised all the submodules recureively.

Next, run Protobuild.exe in /Other/libs/FSOMonoGame/ - this will initialize the Monogame projects used by the FreeSO solution.

The version in the Code/ directory should already come with all the libraries required to build, and Protobuild already run.

Simply open Code/TSOClient/FreeSO.sln in Visual Studio. Set your build target to FSO.IDE, and hit Start. The game should build and start as normal.

Appendix C

Classification of work that is part of this Project

Since Volcanic has been built directly into the FreeSO code-base, it is rather hard to decipher at a glance which parts of the whole solution were made within the scope of this project. Fortunately, most of the code is split into its own project, FSO.IDE, and the GitHub commit log can help you decipher what changes were made for the purposes of the IDE.

Here is a full list of changes which are definitely part of the IDE. Any elements not listed here, even if they have some relation to the IDE or a single line of code changed, can be ignored for any kind of formal marking.

- ALL code files in the FSO.IDE project.
- FSO.Client.Debug.IDEHook, some changes to main path in Program.cs.
- FSO.Content: ChangeManager.cs, ResAction.cs, extensive changes to WorldObjectProvider.cs
- FSO.Files:

  ALL Write() methods in chunk formats, as well as changes to support it

  PiffEncoder.cs, PIFF.cs, PIFFRegistry.cs, IffRuntimeInfo.cs, ChunkRuntimeInfo.cs, SPR2FrameEncoder.cs

  Extensive changes to IffFile.cs to allow dynamic add/remove of chunks as well as change tracking, save to file, revert, patch.
- FSO.LotView: Non-specific changes to World2D to allow render to RenderTarget2D
- FSO.SimAntics:

  Changes to VMThread to support breakpoints and live debug. Changes to VM to signal to UI when IDE events occur.

  Save methods for ALL primitives. Changed fields to C# reflectable parameters.
- SimplePaletteQuantizer: included for use by SPR2 encoder.

Here is a link to the FreeSO/Volcanic GitHub. Relevant changes will be made by me, RHY3756547, and committed with the tag [IDE], along with a description of what was changed:

https://github.com/RHY3756547/FreeSO/commits/master

Appendix D

Survey Questions

Following this page is a printout of all of the questions appearing in the User Evaluation survey. Though participants filled it in electronically, this printout contains all of the same questions and images presented to the user.

Volcanic IDE Evaluation Survey

First of all, thanks for participating in the evaluation! Your feedback will greatly benefit the future development of the IDE, and, of course, give me some real opinions to write about in my dissertation.

The whole process will only take 5-10 minutes, and generally consist of multiple choice questions. All text feedback is optional, though I'd appreciate it if you express any concerns/praise through them if you have any!

There are a few unique sections to get through:
- About You
- Object Browser
- Object Window
- String Editor
- Tree Table Editor
- BHAV Editor
- DGRP Editor
- Overview

Most of these will involve direct comparisons between Edith, Codex, Transmogrifier and Volcanic, with pictures to aid your decisions. If you have not used the tools in question, just base your response on the screenshots provided.

Thanks!
Rhys.

*Required

About You

A few questions just to establish who the participants of the evaluation are. The survey is entirely anonymous apart from this section!

1. **Which of the following tools have you used before?** *
Tick all that apply.

- [ ] The Sims
- [ ] The Sims Online
- [ ] Codex (BHAV editor for TS1)
- [ ] Edith (Official BHAV editor)
- [ ] The Sims Transmogrifier (object recolour)
- [ ] Iff Pencil

2. **Do you have a background in Programming?** *
Mark only one oval.

	1	2	3	4	5	
No Experience	◯	◯	◯	◯	◯	Very Experienced

3. **Do you have a background in The Sims Modding?** *
Mark only one oval.

	1	2	3	4	5	
No Experience	◯	◯	◯	◯	◯	Very Experienced

4. **Do you have a background in SimAntics Programming?** *
Mark only one oval.

	1	2	3	4	5	
No Experience	◯	◯	◯	◯	◯	Very Experienced

5. **Why have you competed the evaluation? (Check all that apply)** *
Tick all that apply.

- [ ] Wish to create new objects
- [ ] Wish to translate existing objects
- [ ] Wish to recolour existing objects
- [ ] Wish to hack/modify existing objects
- [ ] Interested in The Sims gameplay behind the scenes
- [ ] For fun
- [ ] For the offered event object
- [ ] Other: .....

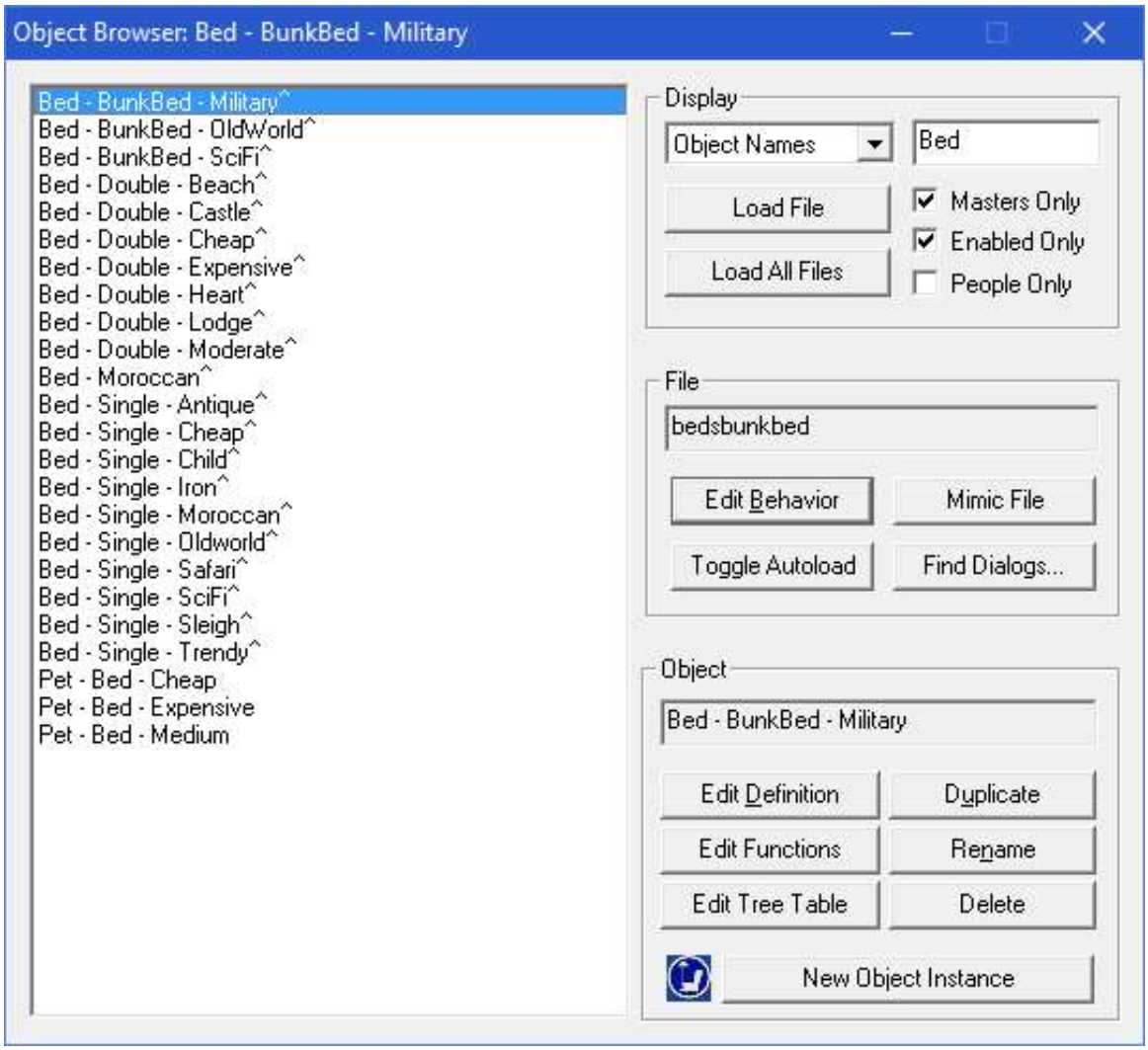
6. **What is your most important reason for completing the evaluation?** *
Mark only one oval.

- ◯ Wish to create new objects
- ◯ Wish to translate existing objects
- ◯ Wish to recolour existing objects
- ◯ Wish to hack/modify existing objects
- ◯ Interested in The Sims gameplay behind the scenes
- ◯ For fun
- ◯ For the offered event object
- ◯ Other reason specified above

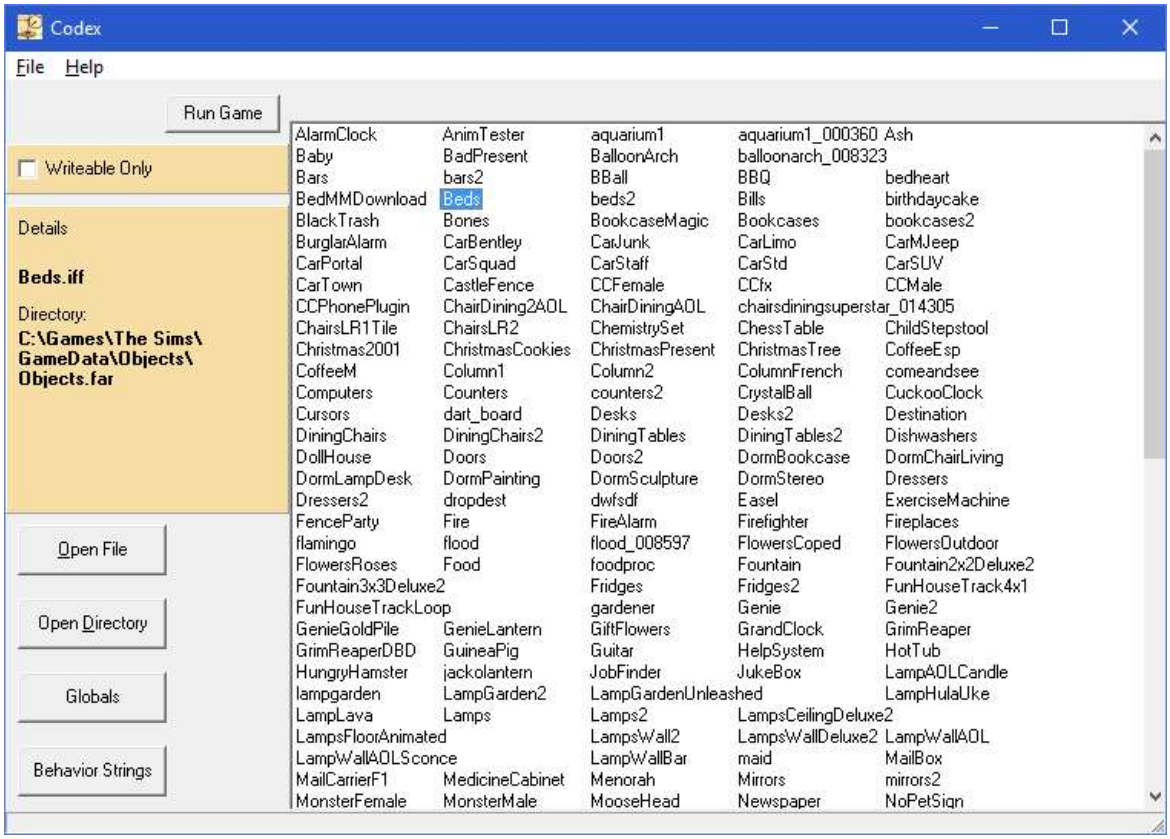
Object Browser

Questions specific to the Volcanic IDE object browser.

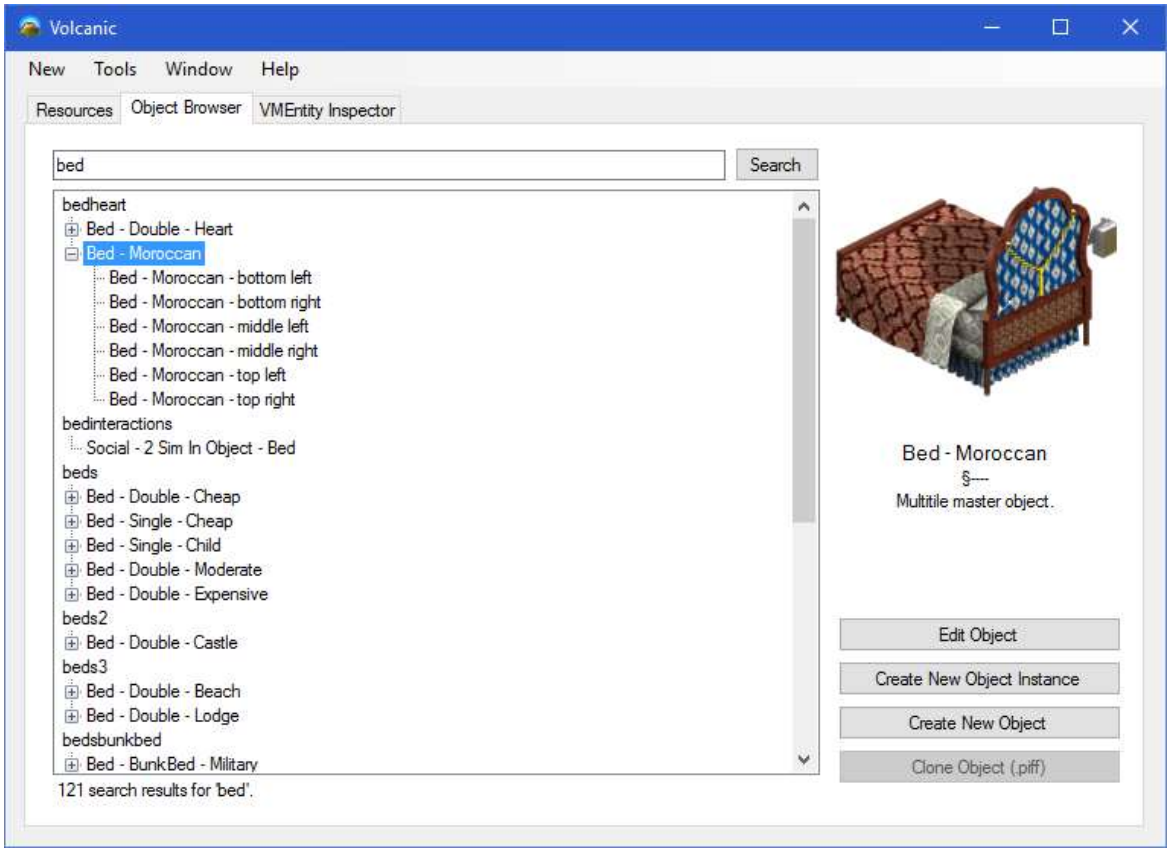
Object Browser in Edith



Object Browser in Codex



Object Browser in Volcanic



7. **Which tool looks easiest to use?** *
Mark only one oval.

- Edith
- Codex
- Volcanic

8. **Which tool looks most powerful?** *
Mark only one oval.

- Edith
- Codex
- Volcanic

9. **Which tool do you think would be most effective at creating objects for a familiarised user?**
Mark only one oval.

- Edith
- Codex
- Volcanic

10. **How easy was it to find the objects you were looking for?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Hard											Easy

11. **Rate how you think these features affect the usability of Volcanic in comparison:** *
Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
Object thumbnail preview					
Search using both OBJD and iff names					
Objects grouped by source iff file					
Seeing all objects in a Multitile Group					
Appearance					

12. **How important do you think the development of these future features is?** *
Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Search using catalog names					
Search by primitives used					

13. **(Optional) Suggest any future features you would like to see.**

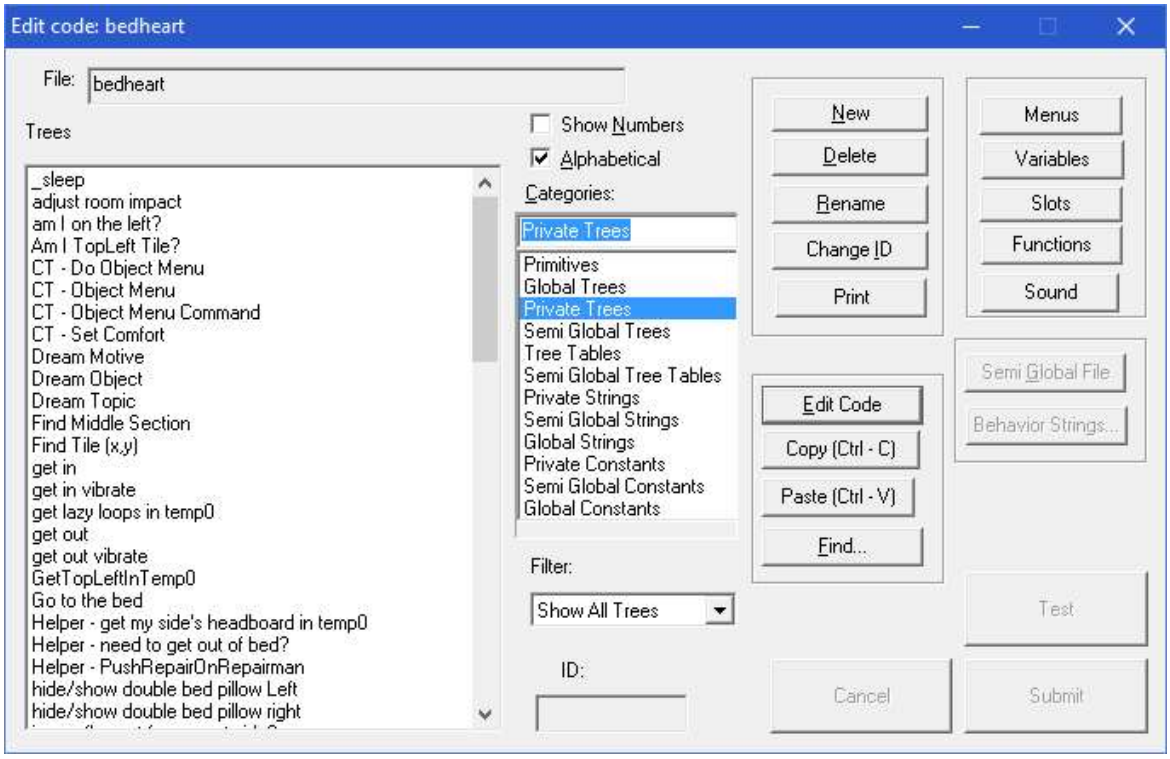
14. **(Optional) What did you like most about the Object Browser?**

15. **(Optional) What problems did you encounter?**

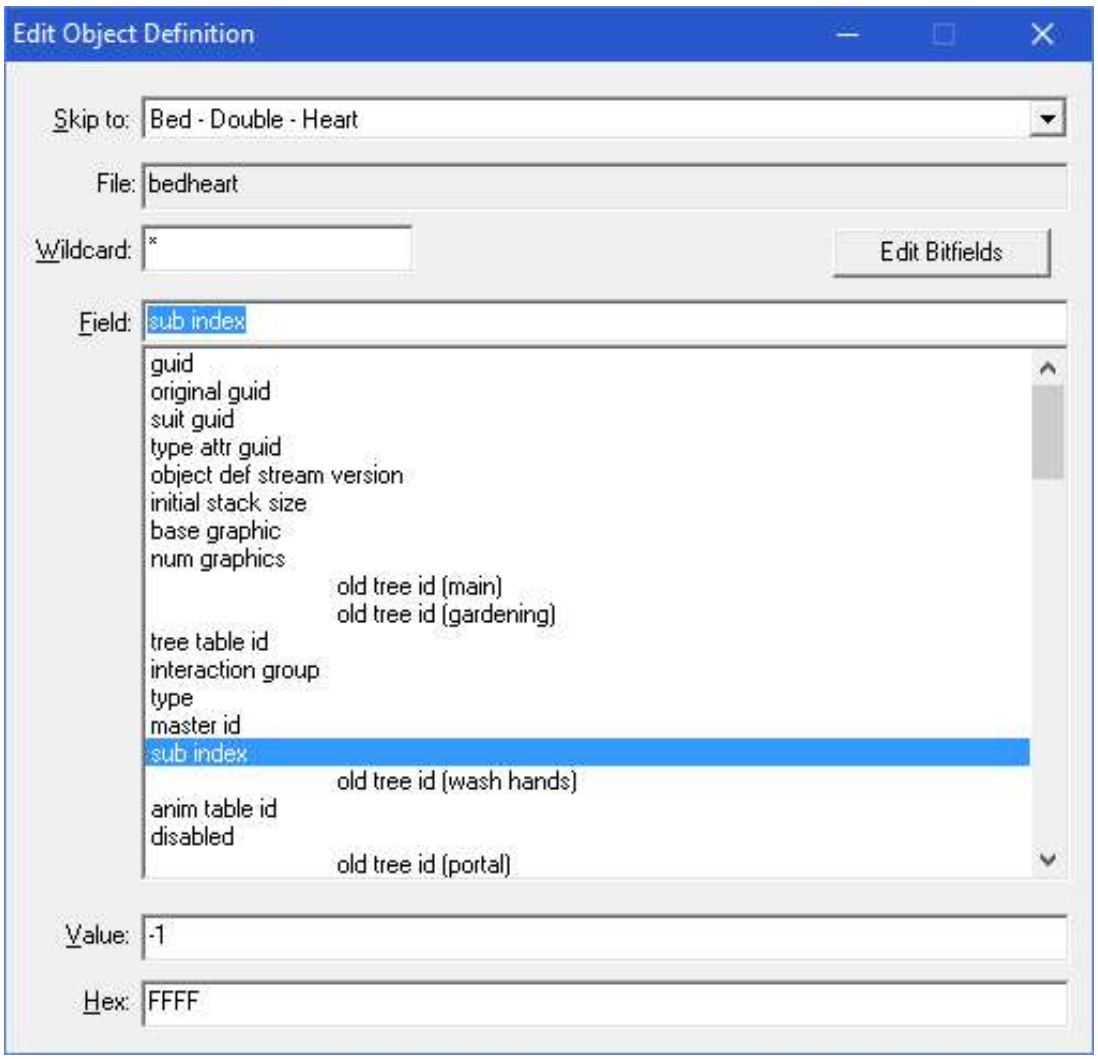
Object Window

Questions specific to the object window, and resource browser.

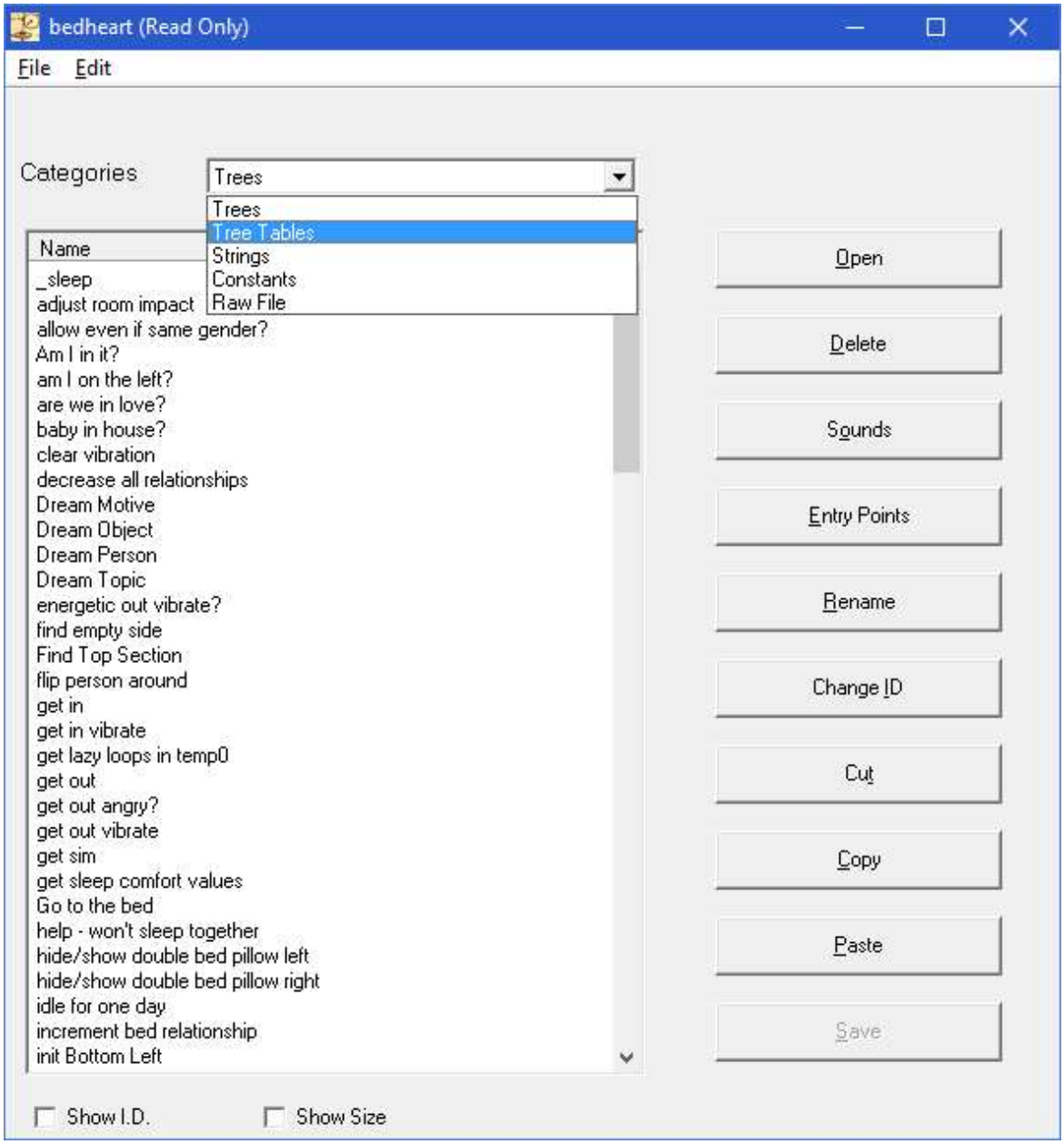
Object Window in Edith



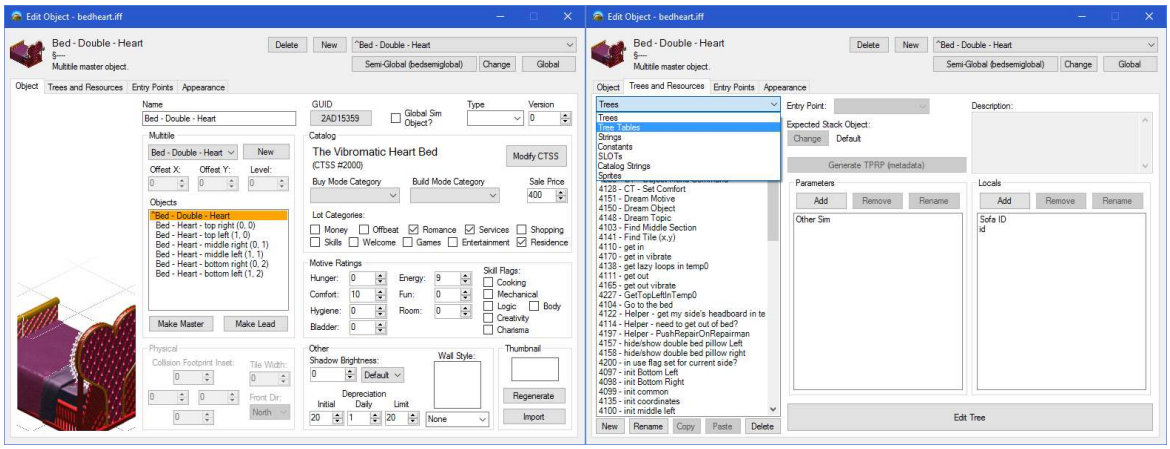
OBJD editor in Edith



Object Window in Codex



Object Window (with OBJD edit) in Volcanic



16. **Which tool looks easiest to use?** *
Mark only one oval.

- Edith
- Codex
- Volcanic

17. **Which tool looks most powerful?** *
Mark only one oval.

- Edith
- Codex
- Volcanic

18. **Which tool do you think would be most effective at creating objects for a familiarised user?** *
Mark only one oval.

- Edith
- Codex
- Volcanic

19. **Do you think the Object Definition editor in the first tab is too complex in its current state? Rate your thoughts on this from 1-10.** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Simple											Complex

20. **How would you rate the current configuration of tabs in the Object Window?** *
Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Good

21. **How would you change this configuration, if at all?**

.....

.....

.....

.....

.....

22. **Rate how you think these features affect the usability of Volcanic in comparison:** *
Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
Object thumbnail preview					
Everything in one place					
Resorurce editor alongside the resource list					
Create/remove objects in the selected file					

23. **How important do you think the development of these future features is?** *
Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
"File Options" page for export to .iff					
Import of resources like BHAVS.					
Export of resources like BHAVs.					
Copy/Paste of resources like BHAVs.					
Cleanup of definition editor					

24. **(Optional) Suggest any future features you would like to see.**

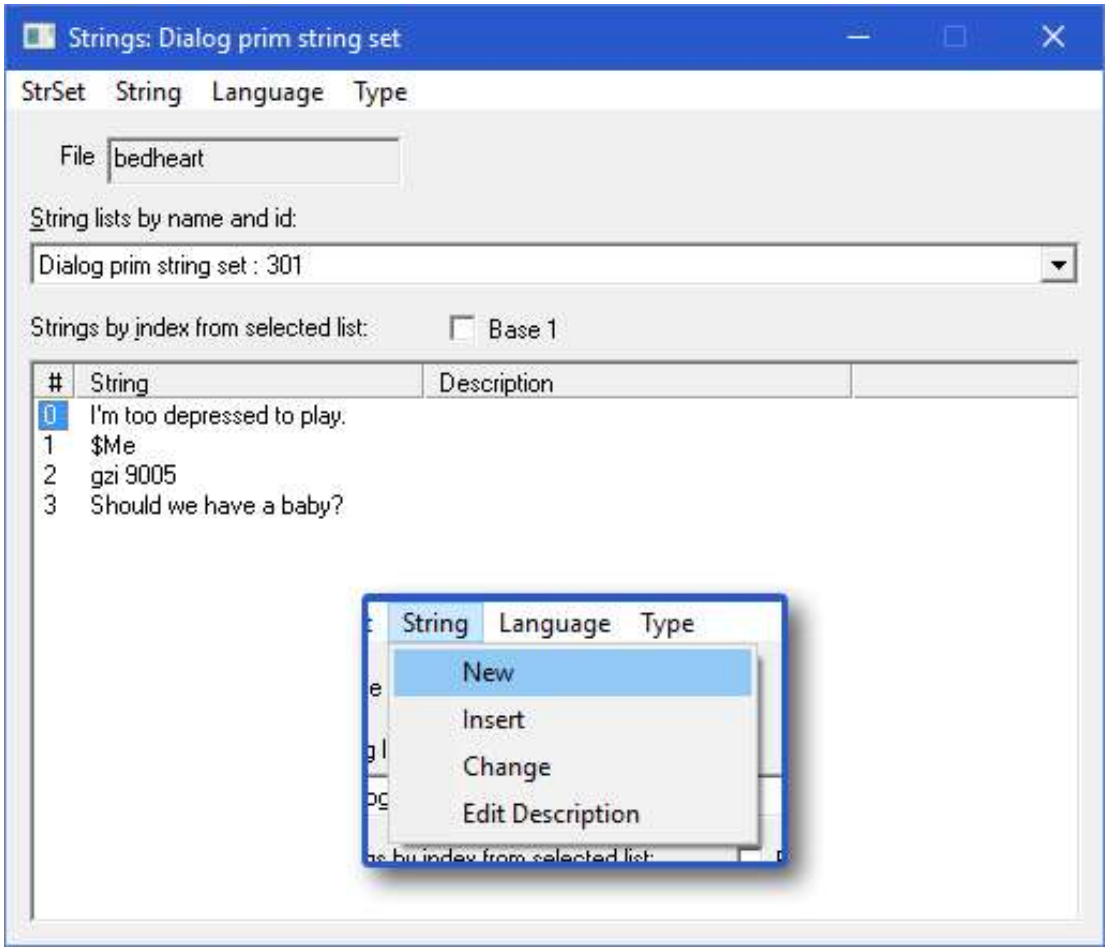
25. **(Optional) What did you like most about the Object Window?**

26. **(Optional) What problems did you encounter?**

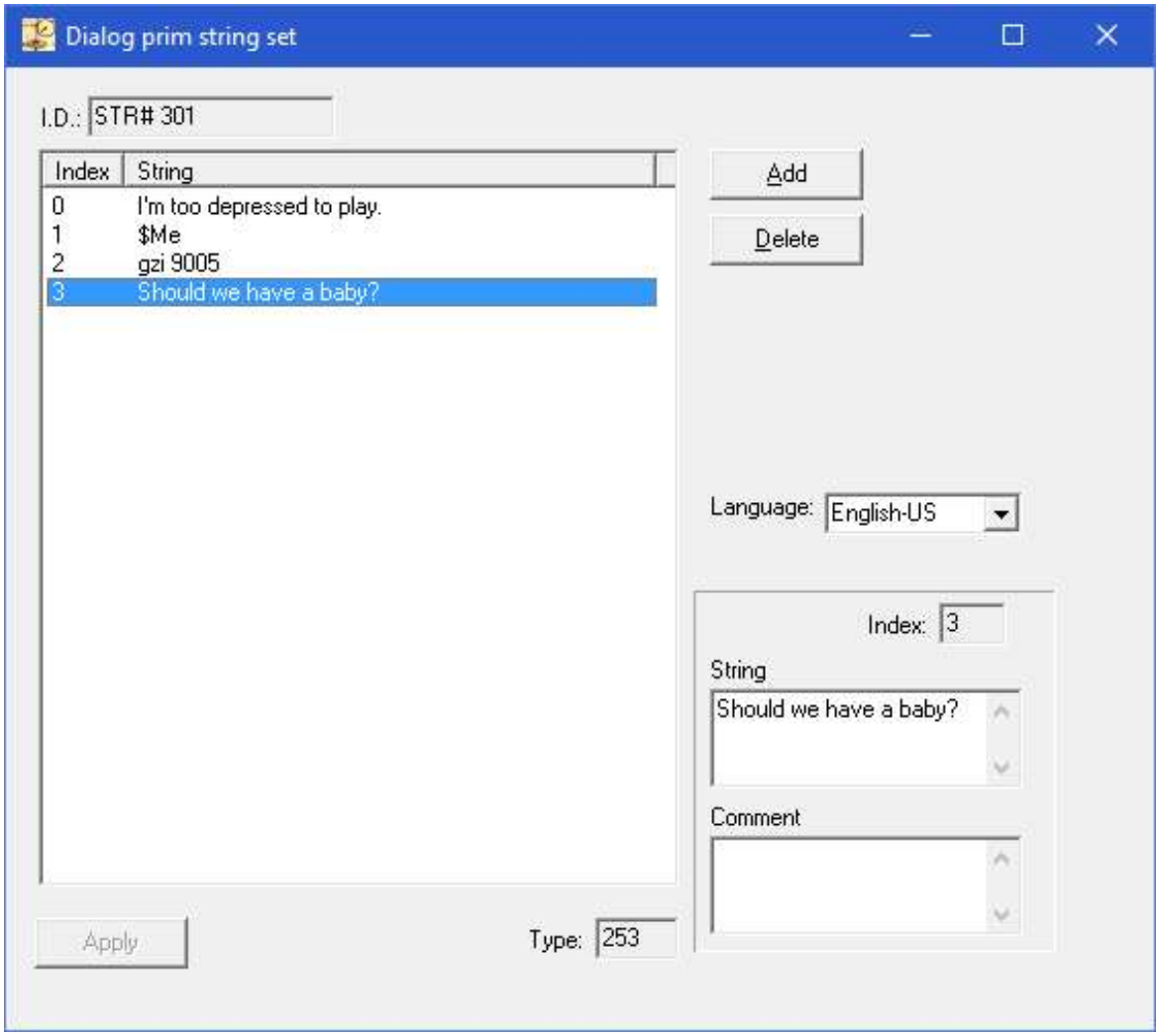
String Editor

Questions specific to the STR resource editor, typically used for translations, making dialogs and catalog strings.

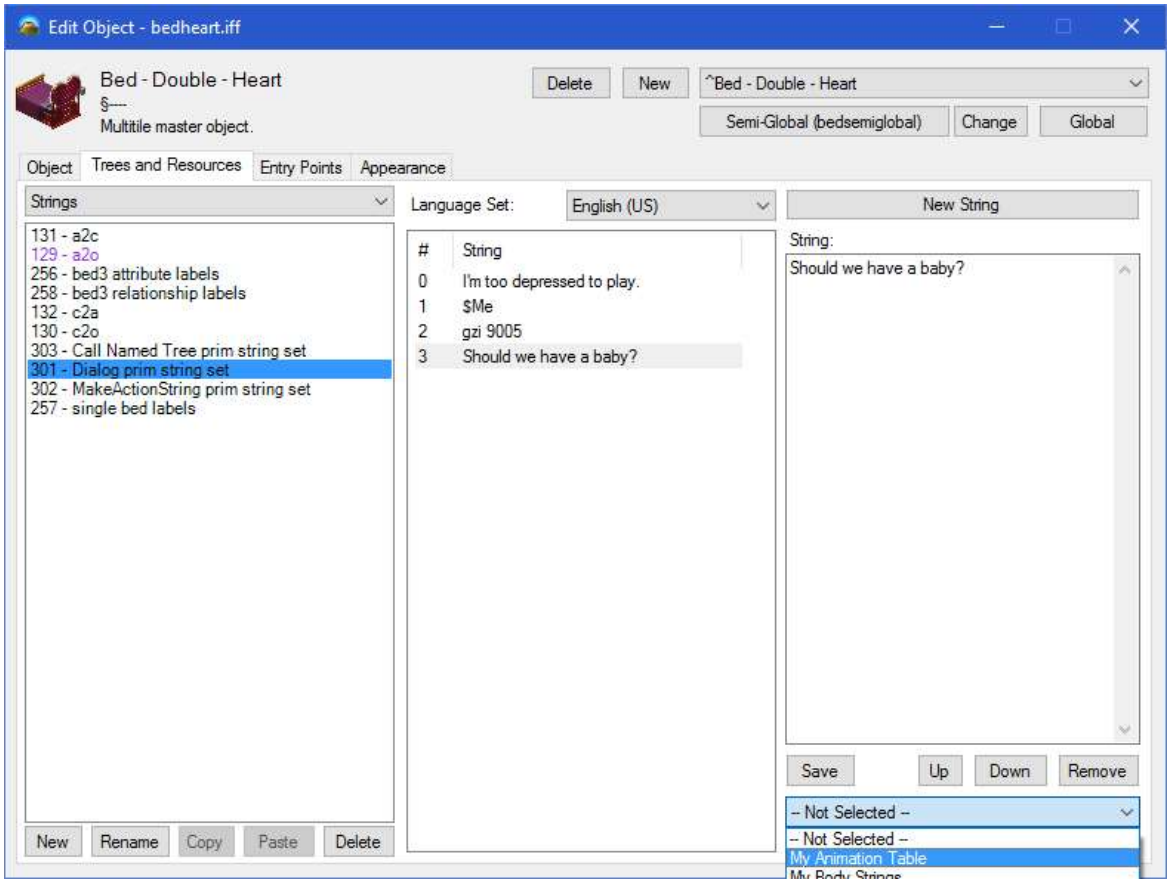
String Editor in Edith



String Editor in Codex



String Editor in Volcanic



27. **Which tool looks easiest to use?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

28. **Which tool looks most powerful?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

29. **Which tool do you think would be most effective at creating objects for a familiarised user?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

30. **Rate how you think these features affect the usability of Volcanic in comparison:** *

Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
Language Set selection					
Automatic creation of Language Sets on switch					
Add/Remove with buttons vs top menu					
Unified reordering across Language Sets					

31. **How important do you think the development of these future features is?** *

Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Comments alongside strings					
Autosave or Prompt on no save					
Custom editor for Catalog Strings					
Remember Lang Set used last					

32. **(Optional) Suggest any future features you would like to see.**

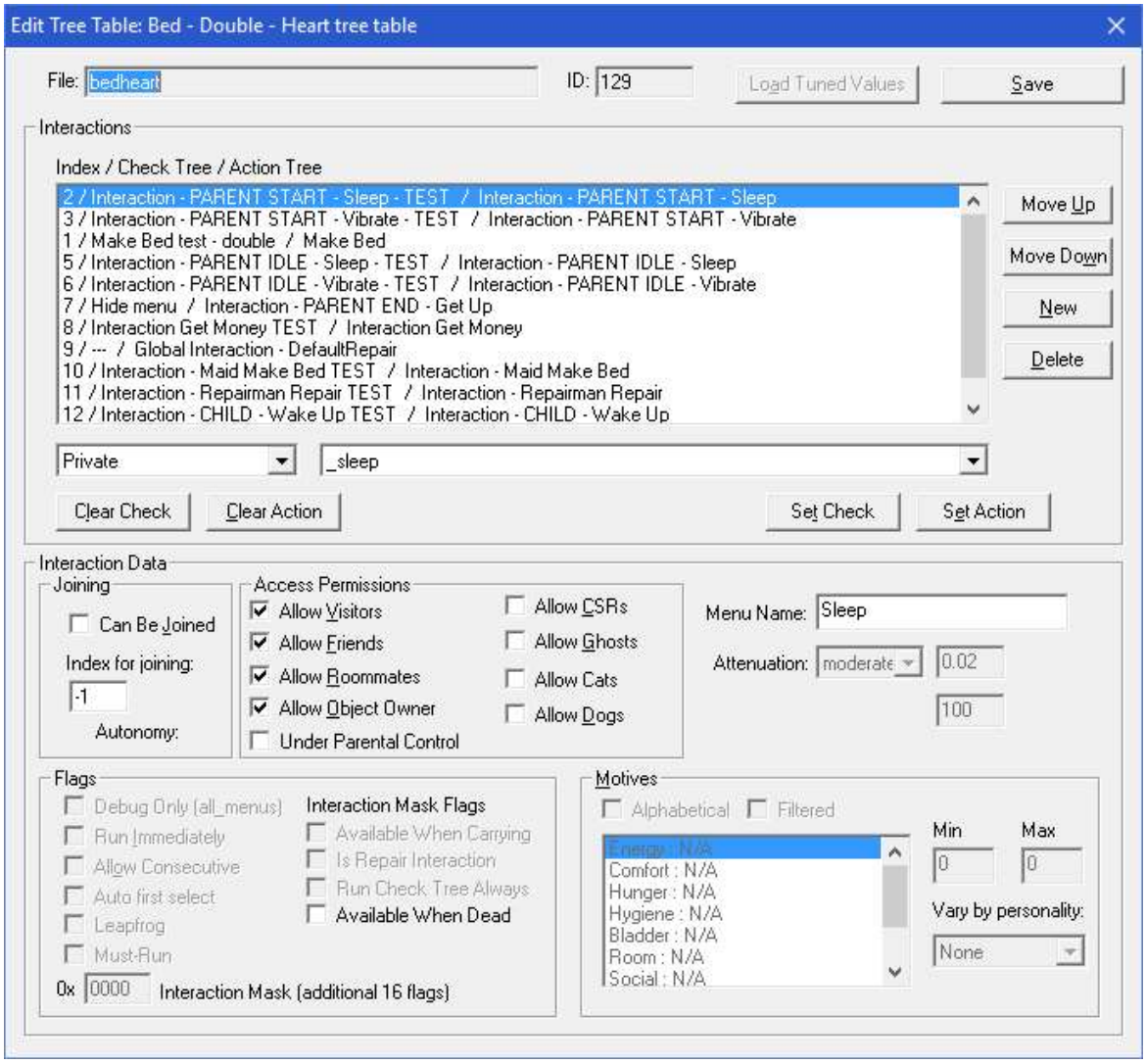
33. **(Optional) What did you like most about the String Editor?**

34. **(Optional) What problems did you encounter?**

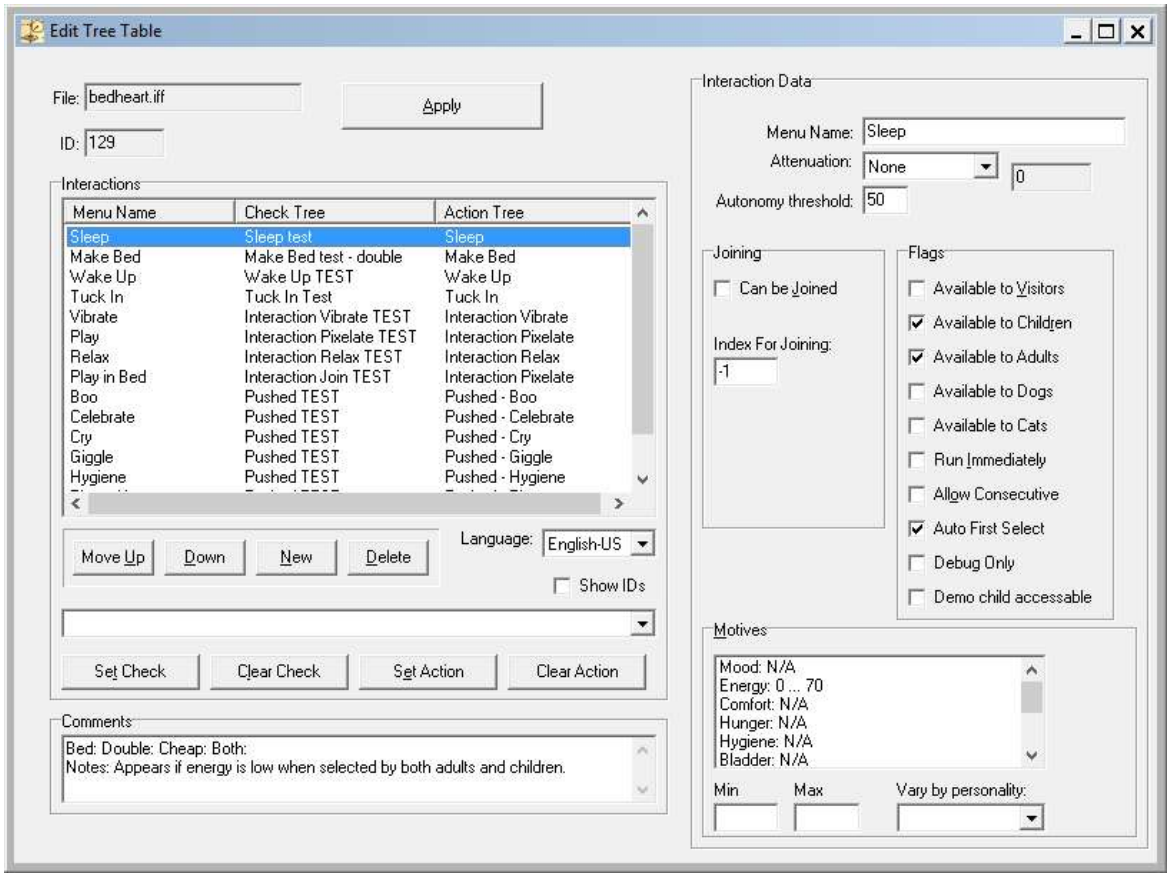
Tree Table Editor

Tree Tables define what appears in the Pie Menu for any object. You should have seen this resource type in any of the "Edit functionality" tasks. If not, just go by their looks.

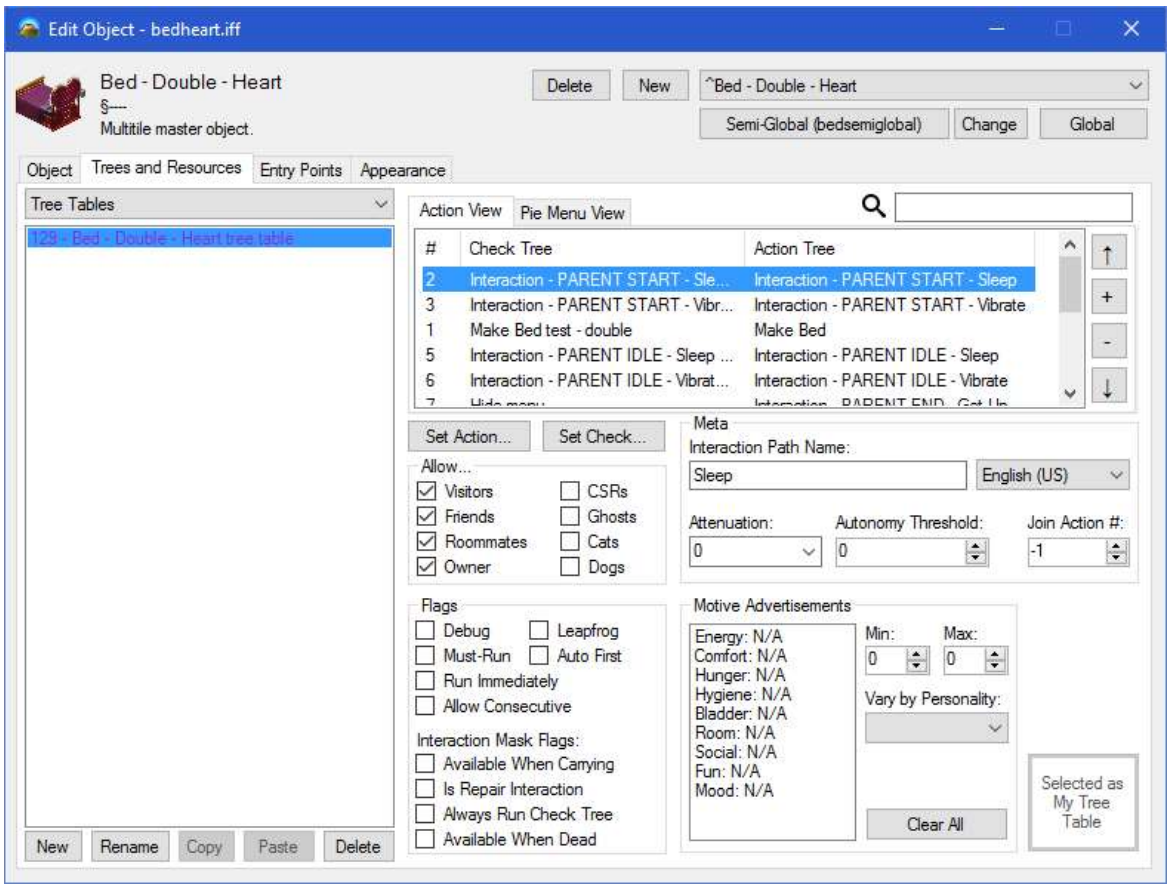
TTAB Editor in Edith



TTAB Editor in Codex



TTAB Editor in Volcanic



35. **Which tool looks easiest to use?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

36. **Which tool looks most powerful?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

37. **Which tool do you think would be most effective at creating objects for a familiarised user?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

38. **Rate how you think these features affect the usability of Volcanic in comparison:** *
Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
Smaller Layout					
"Pie Menu View"					
Language Set switch for Names					
Hiding/Auto edit of TTAs resource					
Appearance					

39. **How important do you think the development of these future features is?** *
Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Comments for interactions					
Revised motive ad display					
Better utilization of space					

40. **(Optional) Suggest any future features you would like to see.**

41. **(Optional) What did you like most about the Tree Table editor?**

42. **(Optional) What problems did you encounter?**

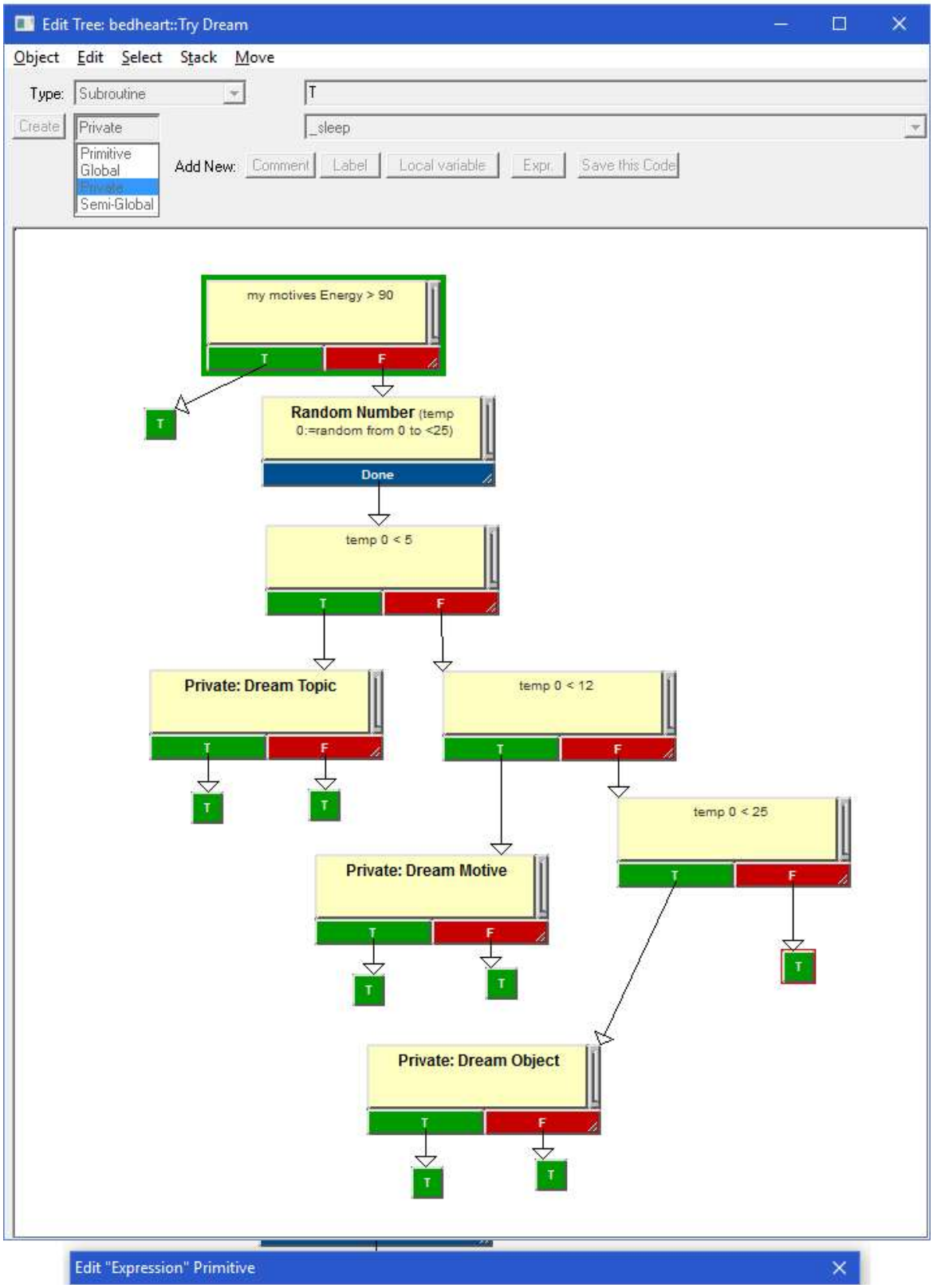
BHAV Editor

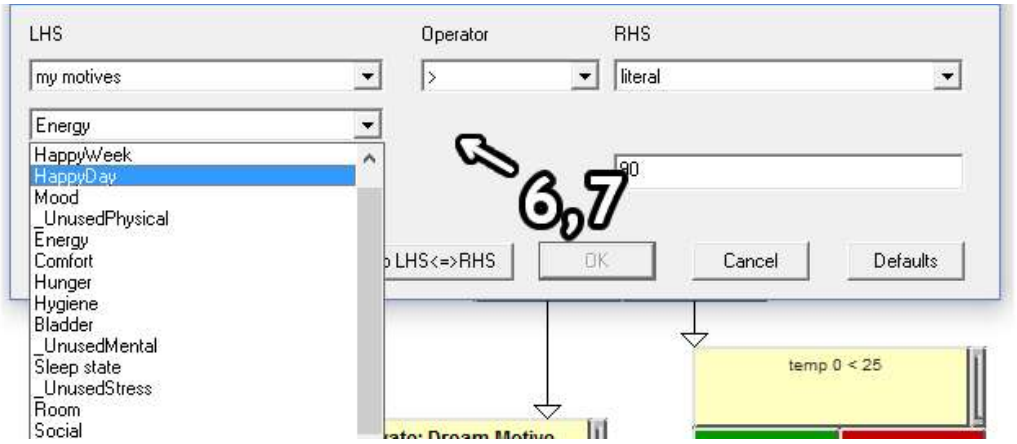
The main show! The comparison images this time will be labelled with numbers, to point out areas of interest. Please take a look at each area in the screenshots for the ratings!

BHAV Editor in Edith

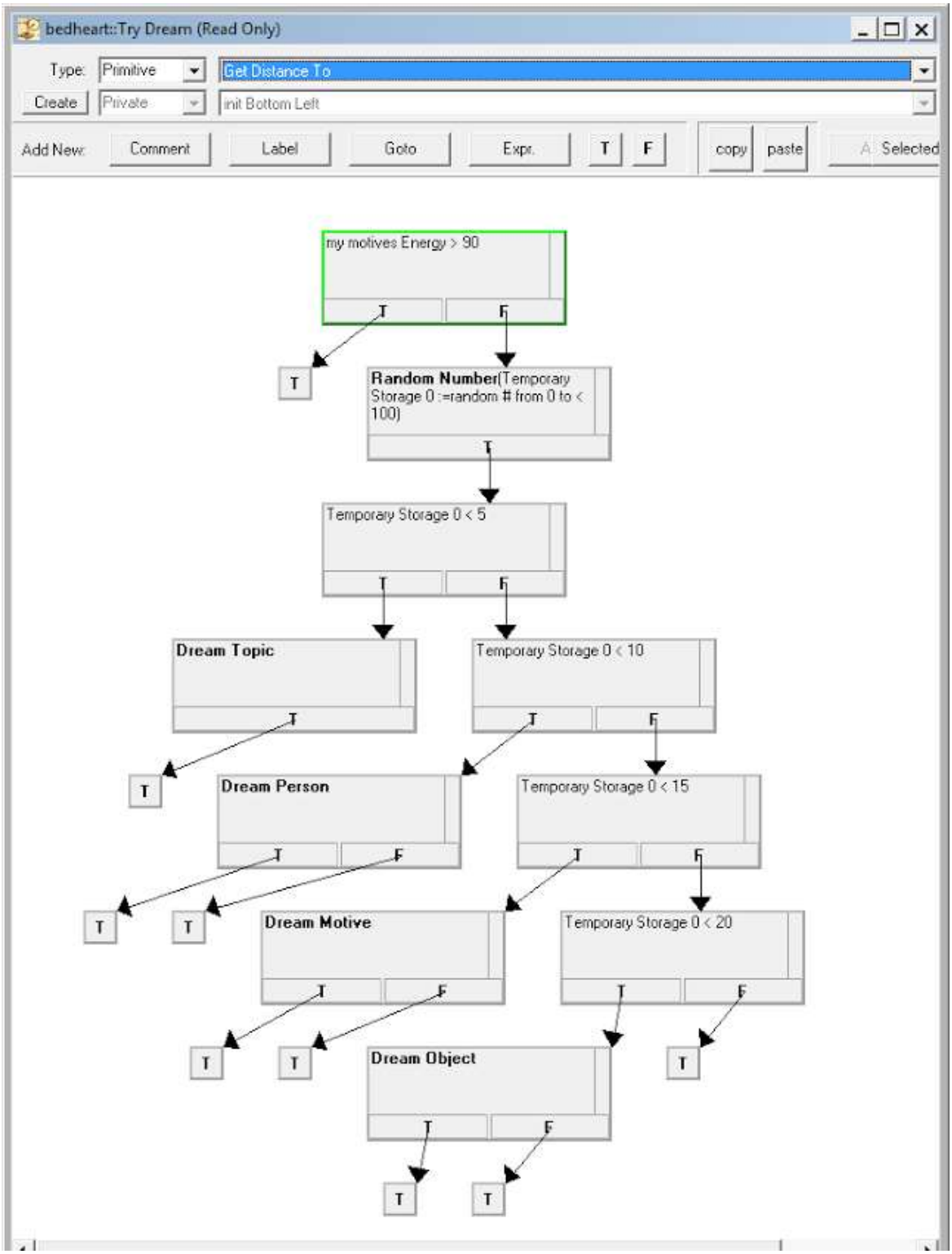
Default:

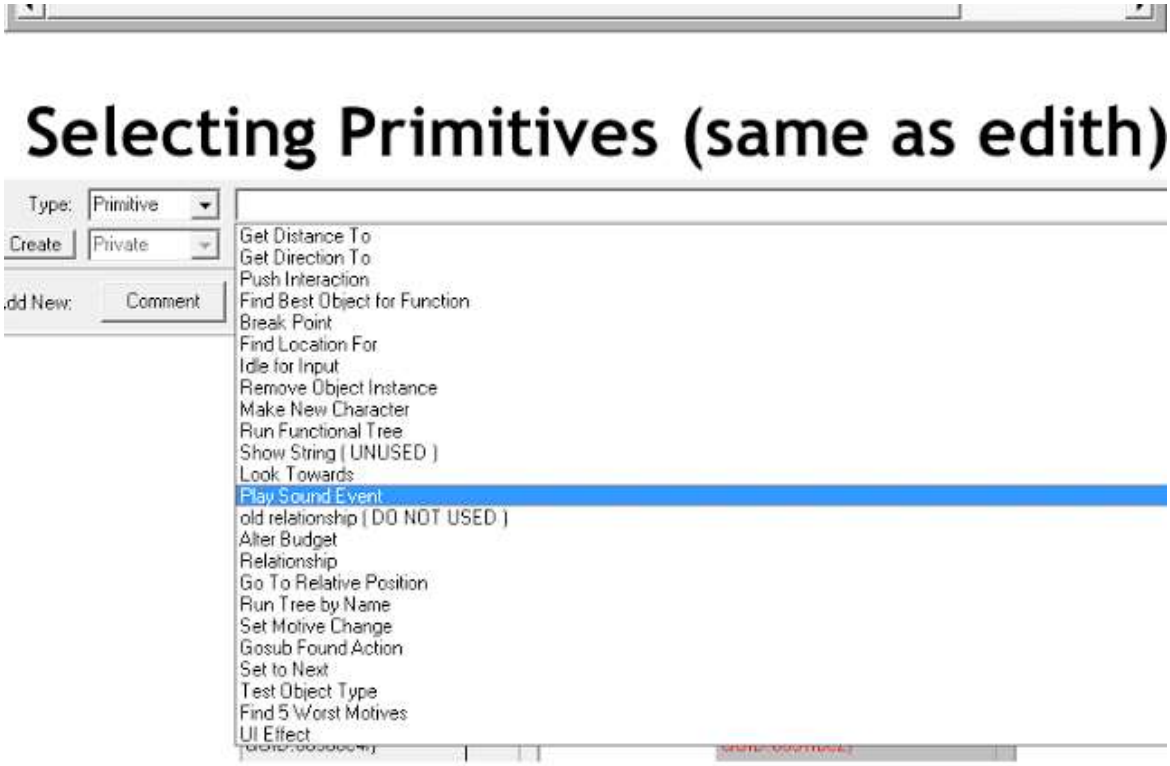
Manually moved:

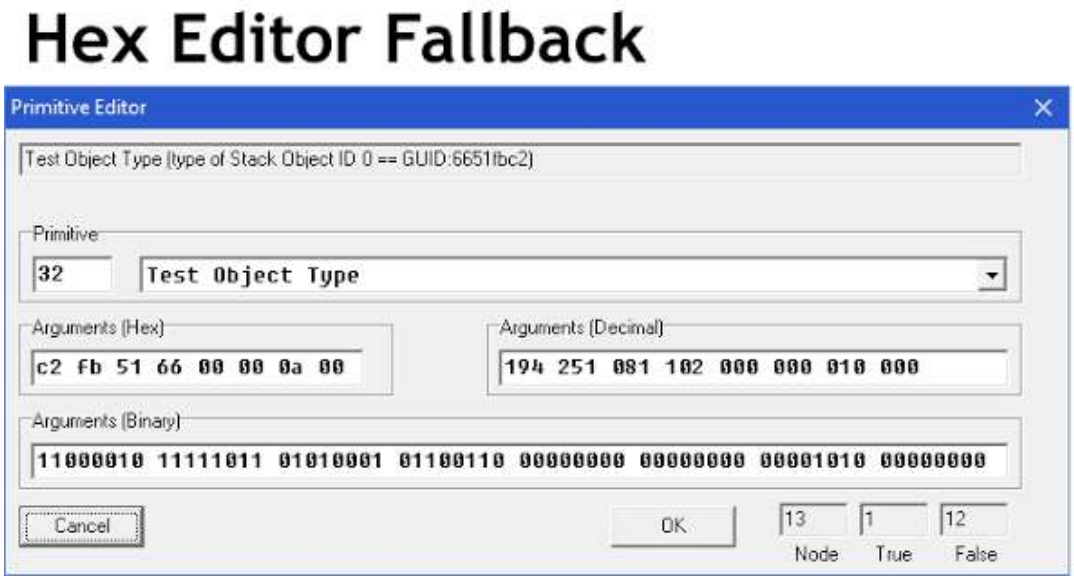




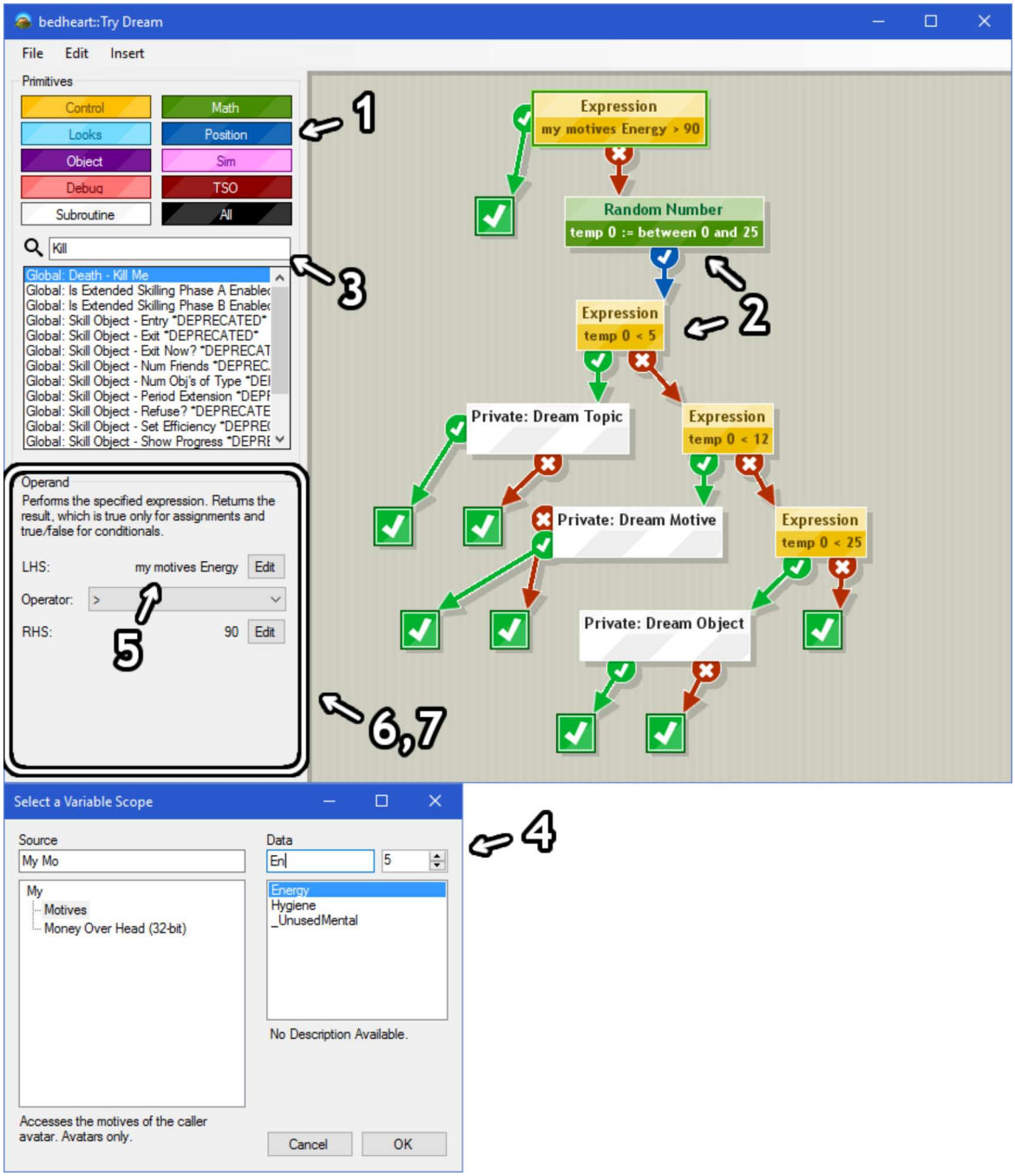
BHAV Editor in Codex







BHAV Editor in Volcanic



43. **Which tool looks easiest to use?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

44. **Which tool looks most powerful?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

45. **Which tool do you think would be most effective at creating objects for a familiarised user?** *

Mark only one oval.

- Edith
- Codex
- Volcanic

46. **Rate how you think these features affect the usability of Volcanic in comparison (see screenshots for numbered references:** *

Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
(1) Grouping Primitives by function					
(2) Colouring of primitive groups					
(3) Primitive selection using Search					
(4) Variable selection using Search					
(5) Determining and Displaying variable names					
(5) Visual operand editor					
(6) Operand edit in same window instead of dialog					
Automatic Tree Layout					
Undo/Redo					
Appearance					
Layout in general					

47. **How useful was the "Breakpoint" functionality in Volcanic for debugging trees?** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Did not use											Very useful

48. **(Optional) When in "Tracer" mode, how useful do you think being able to edit variables on the fly was?**

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Not useful											Useful

49. **How important do you think the development of these future features is?** *
Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Zoom in/out					
BHAV Comments					
Saving Tree Layout					
Labels for sequences of primitives, and "goto" to jump to named sequences.					
Sound selector with preview					
Outfit/Mesh selector with preview					
Multiple selection of primitives					
Copy/Paste of primitives					
Text based language for SimAntics					

50. **(Optional) Suggest any future features you would like to see.**

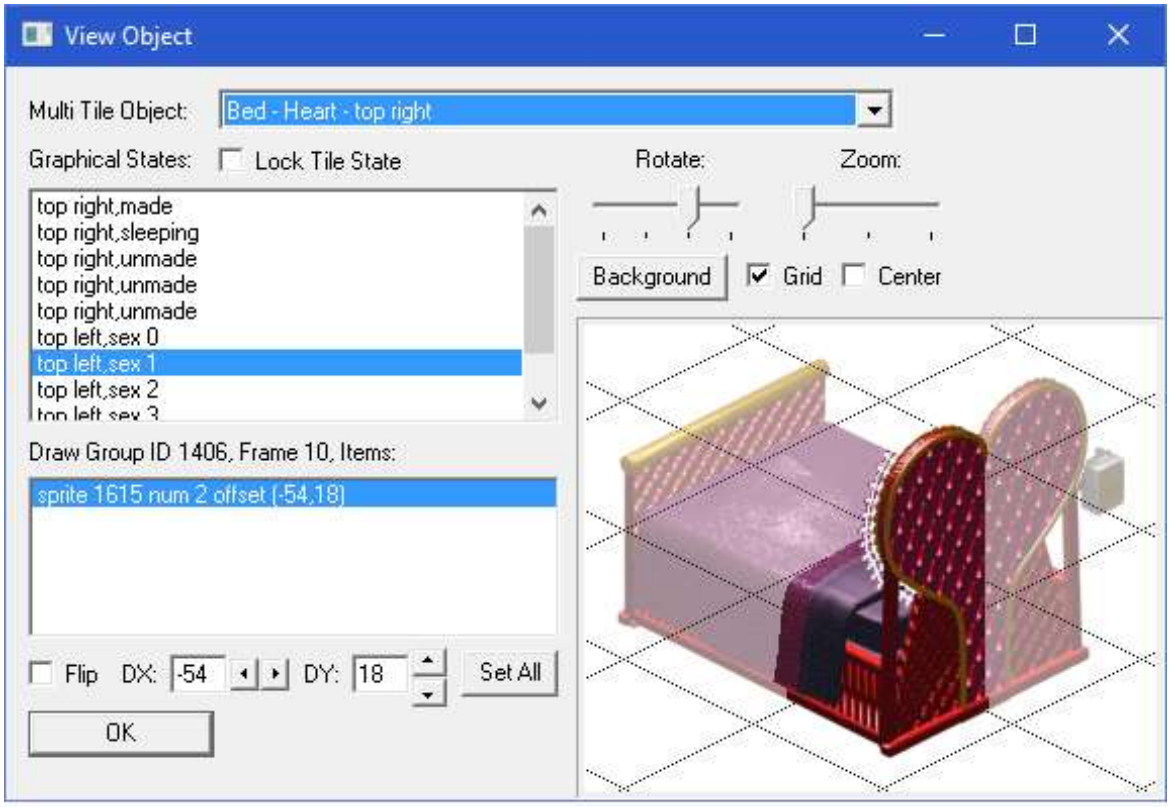
51. **(Optional) What did you like most about the BHAV Editor?**

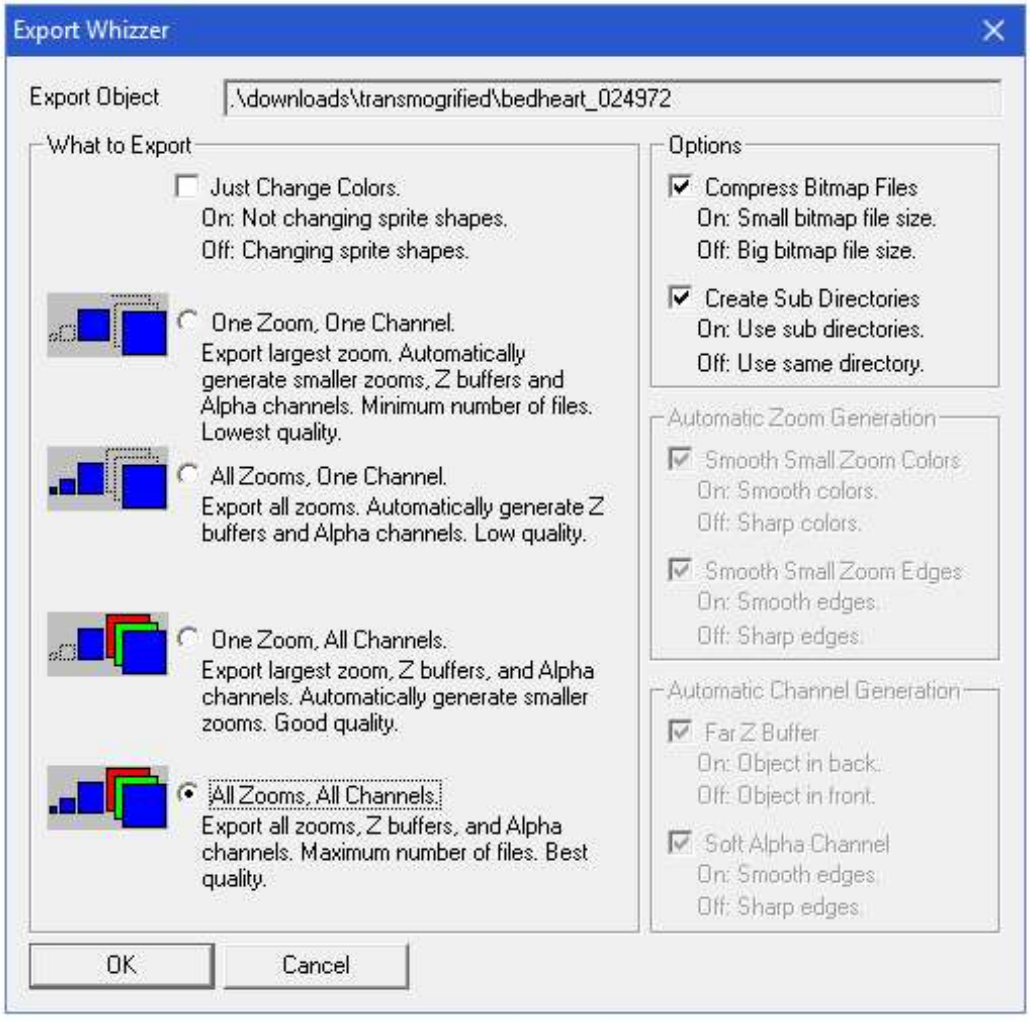
52. **(Optional) What problems did you encounter?**

Graphic Editor

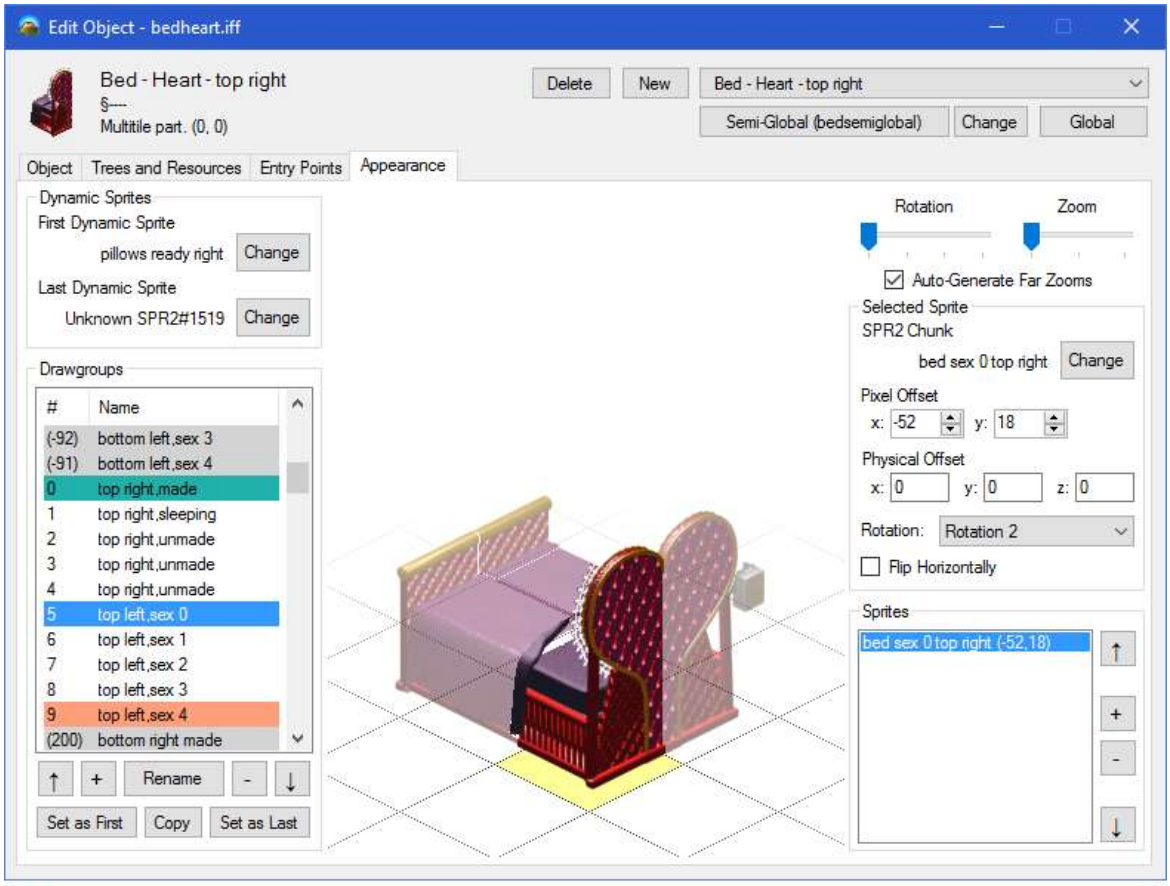
This section covers creating and editing graphics. The only tool to compare with here is Transmogrifier, so questions will be a little simpler.

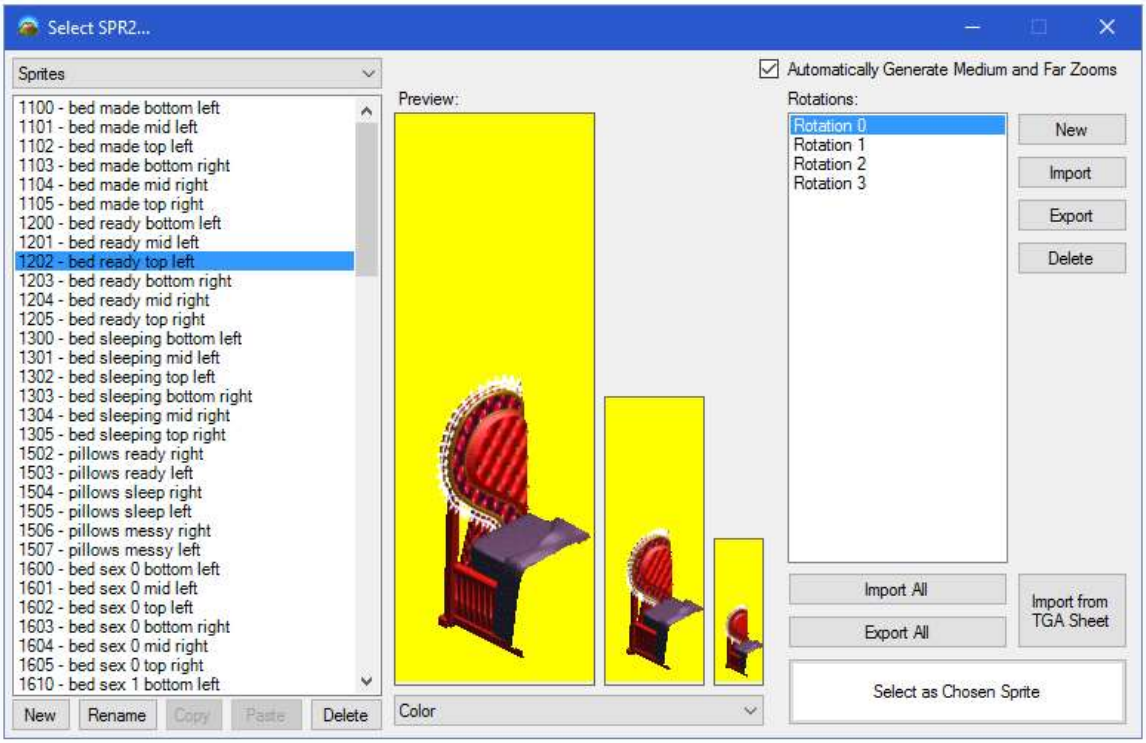
Editing in The Sims Transmogrifier





Editing in Volcanic





53. **Which tool looks easiest to use?** *

Mark only one oval.

- The Sims Transmogrifier
- Volcanic

54. **Which tool looks most powerful?** *
Mark only one oval.

- The Sims Transmogrifier
- Volcanic

55. **Which tool do you think would be most effective at creating objects for a familiarised user?** *
Mark only one oval.

- The Sims Transmogrifier
- Volcanic

56. **Do you think the presence of the Volcanic DGRP editor detracts from the simplicity of changing sprites?** *
Mark only one oval.

- Yes
- No

57. **Rate how you think these features affect the usability of Volcanic in comparison:** *
Mark only one oval per row.

	Much worse	Worse	No effect	Better	Much better
Add/Remove Drawgroups					
Control object's DGRP range					
Import from 3DS Max tga sheet					
Reorder Drawgroups					
Clone existing Drawgroups					
Edit alongside other resources					
Live preview					

58. **How important do you think the development of these future features is?** *
Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Visual selection of sprites with mouse					
Auto-generate rotations					
Import many sprites at once					

59. **(Optional) Suggest any future features you would like to see.**

60. **(Optional) What did you like most about the Graphic Editor?**

61. **(Optional) What problems did you encounter?**

Overview

Wrapping it up! Thanks for sticking with it all this way - you're the best!

62. **Overall, do you think the development of the tool was worth it?** *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Not Worth	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Worth the effort

63. **How much documentation (explaining how to use it) do you think each of these areas requires?** *

Mark only one oval per row.

	None	Some	Moderate	Much	Extensive
BHAV Tree Editor	◯	◯	◯	◯	◯
String editing/translation	◯	◯	◯	◯	◯
Tree Table editing	◯	◯	◯	◯	◯
Object Definition editing	◯	◯	◯	◯	◯
Multitile object management	◯	◯	◯	◯	◯
Change discard/save	◯	◯	◯	◯	◯
Object Browser	◯	◯	◯	◯	◯

64. **How important do you think the development of these future features is?** *

Mark only one oval per row.

	Waste of time	Unimportant	Neutral	Important	Crucial
Animation Editor & Import					
Sound (HIT) Editor & Import					
SLOT resource editor (custom routing positions, table positions)					
The Sims 1 port					
Run without game open					
Documentation and Tutorials					

65. **(Optional) Suggest any future features you would like to see.**

66. **(Optional) Any thoughts, feelings? Please share comments about anything on your mind!**

Thank you!

Thank you for participating in the survey. Make sure to post that you have completed the survey in the Volcanic survey thread, so that I can make sure that you recieve the event object in future. (and so I can make sure everyone has done the survey >:))

Your help is greatly appreciated, and I won't forget it. <3

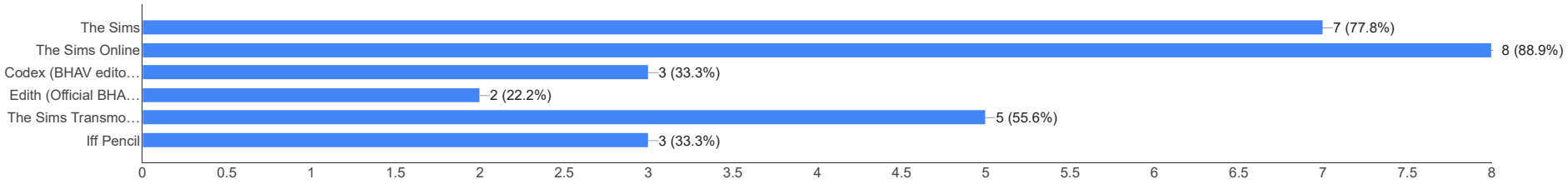
Appendix E

Survey Results

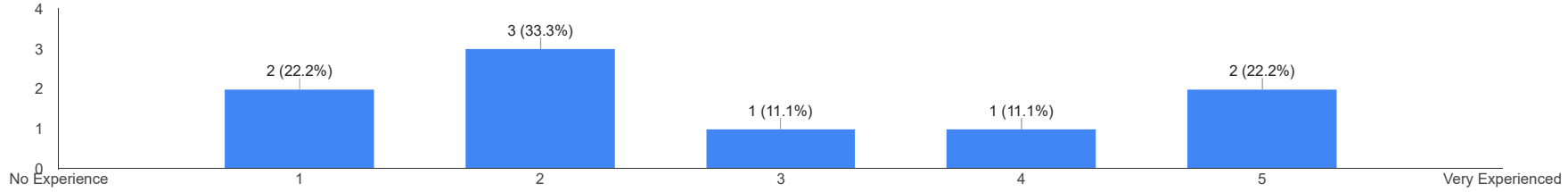
Following this page is a summary of all responses to the User Evaluation survey. Please read the survey itself before reading these, as many of the questions refer directly to the content of the images shown to the user.

About You

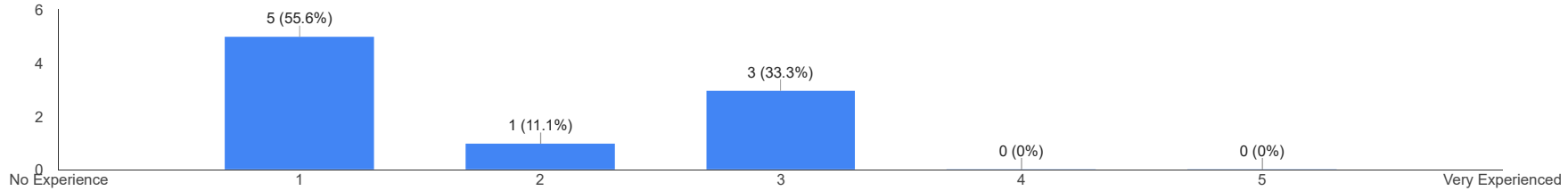
Which of the following tools have you used before? (9 responses)



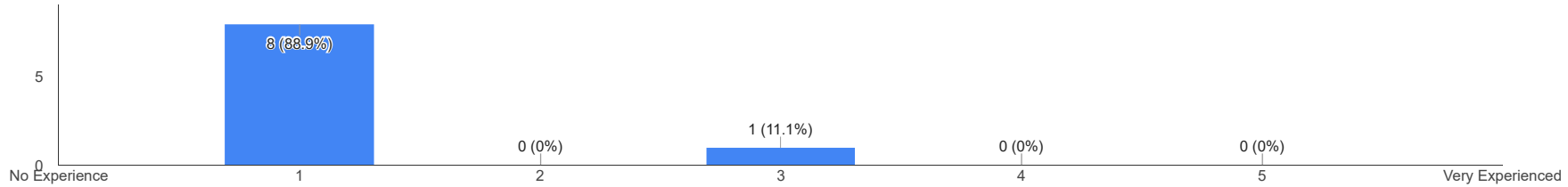
Do you have a background in Programming? (9 responses)



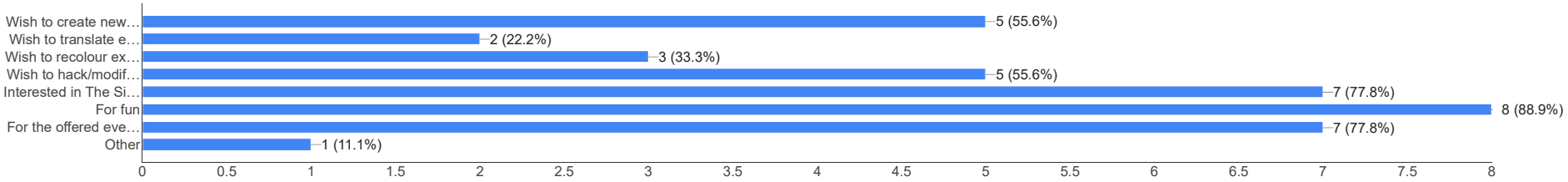
Do you have a background in The Sims Modding? (9 responses)



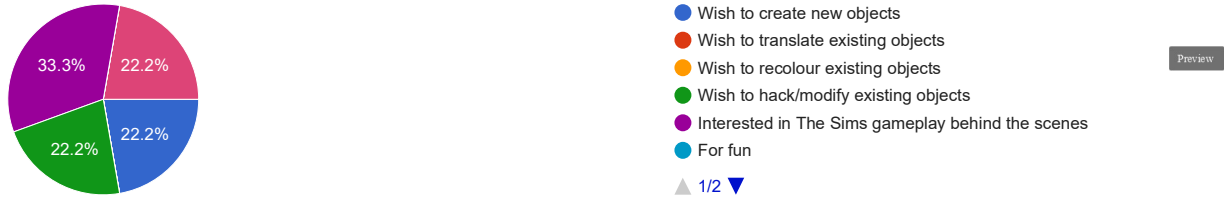
Do you have a background in SimAntics Programming? (9 responses)



Why have you competed the evaluation? (Check all that apply) (9 responses)



What is your most important reason for completing the evaluation? (9 responses)



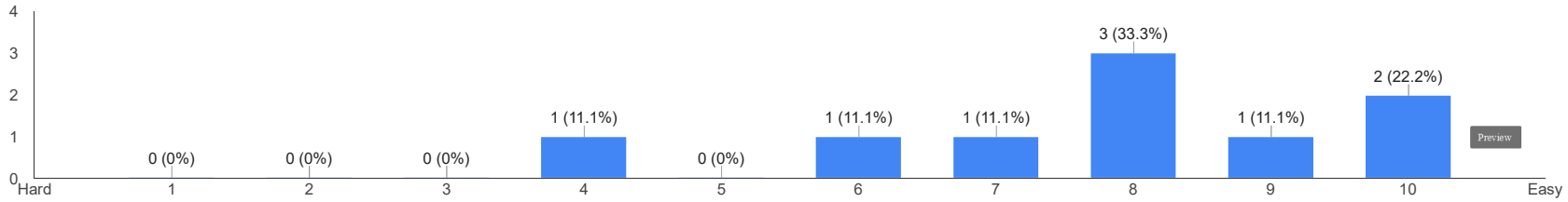
Object Browser

Which tool looks easiest to use? (9 responses)

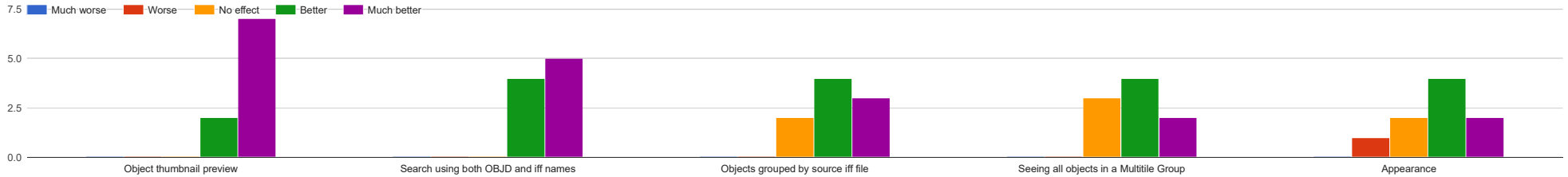
Which tool looks most powerful? (9 responses)

Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

How easy was it to find the objects you were looking for? (9 responses)



Rate how you think these features affect the usability of Volcanic in comparison:



How important do you think the development of these future features is?

(Optional) Suggest any future features you would like to see. (5 responses)

Skins editing .. would be fun , and an innovation Comparing to the Old TSO

Option to search by catalog name, or sort the list by catalog category.

Creation of custom dialog and editing of existinf dialogs and sound effects

A picture/thumbnail next to the object you've selected. Maybe categories so you can look at only tables or only beds.

Catalog preview images when editing strings

(Optional) What did you like most about the Object Browser? (8 responses)

It is quite easy to find the objects

Facility to find some objects

The ease at how easy it is to recall an object. They are scattered in numerous files. To first have to find the exact file an object is. Then to see if the object IS in t here is a nightmare. With it all just being recalled by default is a very helpful to quickly search the object it needs.

Object thumbnail preview

The search feature, grouping objects by iff file.

The powerful search and the thumbnail support

Streamlined and non-confusing interface, easy to start something.

Objects grouped by iff files makes it easy to find groups of objects

(Optional) What problems did you encounter? (6 responses)

Basic problems , such as errors Relating to Palette by sprites..

The primary issue I had with finding the objects was not knowing what I was really looking for. It's not possible to search "Bear statue" as behind the scenes the object is in fact called "Sculpture - Chainsaw". However. I was able to remedy this problem by just recalling how the Maxis team structured the iff, so I knew I had to look in the "sculptures" file and then see which EP it belonged to (3, for House Party, a thing I didn't realize, never knew it came from that expansion pack, thought it was Living Large)

The create new instance button didn't always produce an object if there was no room in the current view of the lot.

Displays in HiDPI or zoomed Modes destroy the layout and interface of the IDE

Some names did not match their catalogs and were quite inconvenient to search for.

Sprite importer/exporter sometimes wouldn't import.

Object Window

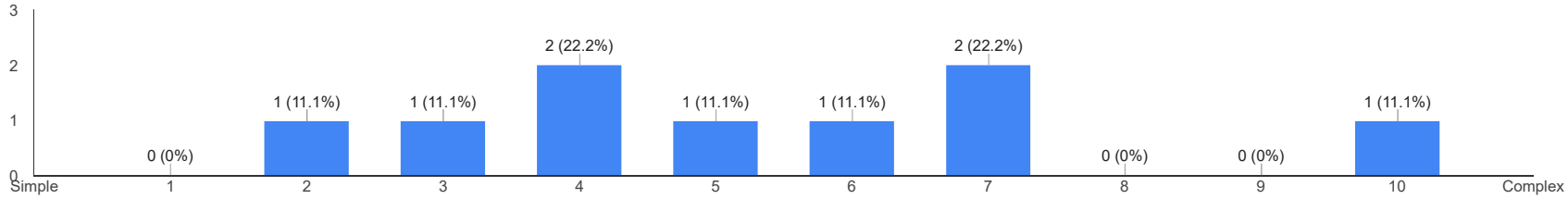
Which tool looks easiest to use? (9 responses)

Which tool looks most powerful? (9 responses)

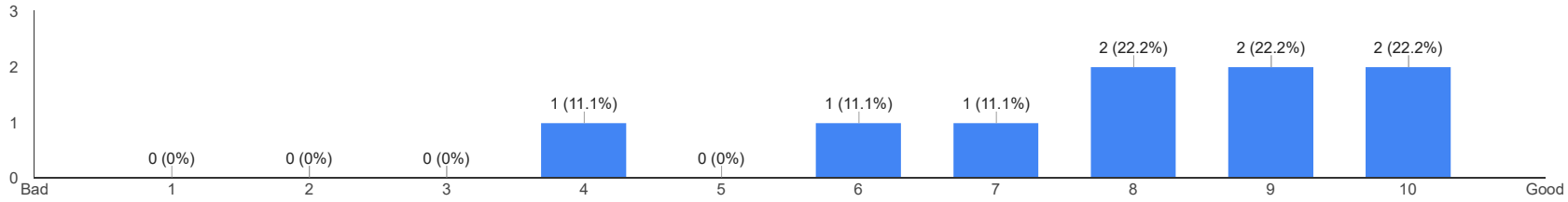
Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

Preview

Do you think the Object Definition editor in the first tab is too complex in its current state? Rate your thoughts on this from 1-10. (9 responses)



How would you rate the current configuration of tabs in the Object Window? (9 responses)



How would you change this configuration, if at all? (8 responses)

no,looks good for me

A little complicated , but over time it becomes easy.

I would probably move "Appearance" to second tab, because it works well with progression from the basic settings in the first tab. while the more advanced options should be later. So a player will first take care of the Basic things such as what category the object is permitted, cost, etc. then move to the appearance section, and then start adding all the functionalities of it.

Sprites tab

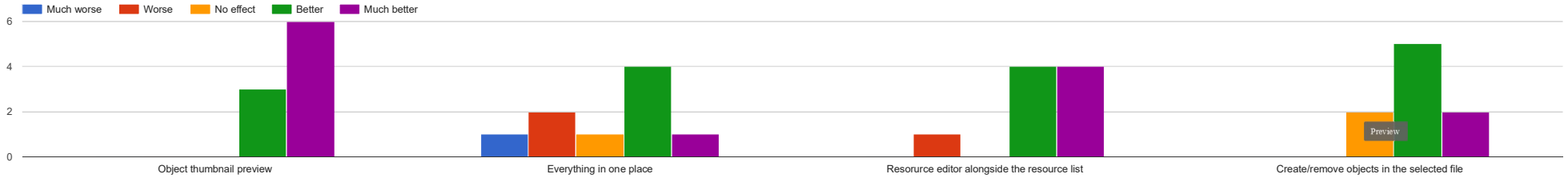
Rearranging the tabs so that Object and Appearance are next to each other

Put the Multitile information to the Appearance tab and make the Object Preview bigger

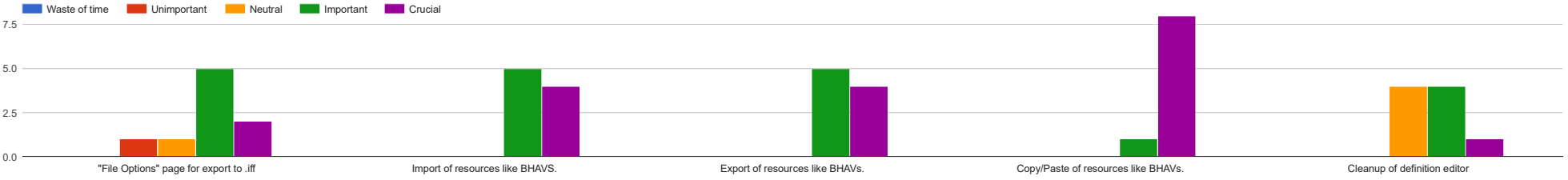
Add the ability to hide tabs so it's not overwhelming to a user.

I like it right now. Haven't had problems finding what I need to.

Rate how you think these features affect the usability of Volcanic in comparison:



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (2 responses)

Copying BHAV's and trees, importing them from other objects to modify.

Being able to change or created Semi-Globlas to creating custom skill objects

(Optional) What did you like most about the Object Window? (5 responses)

It doesn't look intimidating. At first glance you get a basic understanding of what everything is. Edith, Codex and the rest look VERY intimidating and "too advanced" which makes them very unappealing because you may get a sense of "should I even bother?"

Everything is in one place, no hunting or opening multiple windows to find what you want.

The great way flags and stats are presented

It's really easy to change catergories, skills and motive ratings. Also helps you understand multi tile objects.

I like how every value that you can change on that page is labeled with a word that accurately relates to when that value changes

(Optional) What problems did you encounter? (5 responses)

There doesn't seem to be many issues with the actual Object Window. It's straightforward. My only issue maybe the the Chunk IDs can feel scattered. I had few issues with the object not rendering itself and it may have been due to IDs, I had issues with interactions because the ID was Semi-Global and I didn't understand that then (now I do). But most of these are pretty minuscule.
Appereance and sprite editing
Chunk ID's could be automatically generated when adding new resources.
None, didn't use this tab much.
Not really any to be honest. It's easy to see what thing changes what

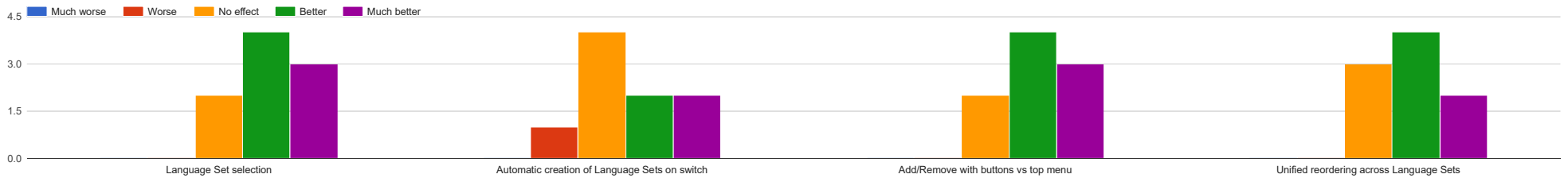
String Editor

Which tool looks easiest to use? (9 responses)

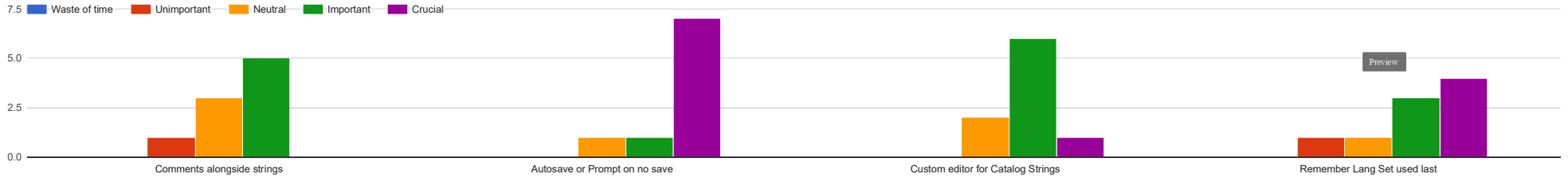
Which tool looks most powerful? (9 responses)

Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

Rate how you think these features affect the usability of Volcanic in comparison:



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (1 response)

Remember the current language your working on and apply it to all resources being edited.

(Optional) What did you like most about the String Editor? (4 responses)

It's very hard. Strings are the easiest thing really. And all three have simple interfaces. In fact, the Strings section of the Object Window is almost distracting (but it needs to be there as Strings don't have their own tab). Again. it's really close "tie" of what looks like the best tool to modify strings.

Easy to add and edit strings.

The easy way to read and edit/translate strings

I could edit almost any strings in the game. This opened up many opportunities for creativity!

(Optional) What problems did you encounter? (4 responses)

There is no way of knowing how strings actually work. I created two strings and had no idea how to actually apply them. My name string worked but my description string didn't.

No prompt to save edited strings before switching to another caused loss of work.

I didn't understand what you do on this tab, also there are weird strings called a2o etc and I don't understand what they do.

Losing work due to clicking out of the box prior to saving work

Tree Table Editor

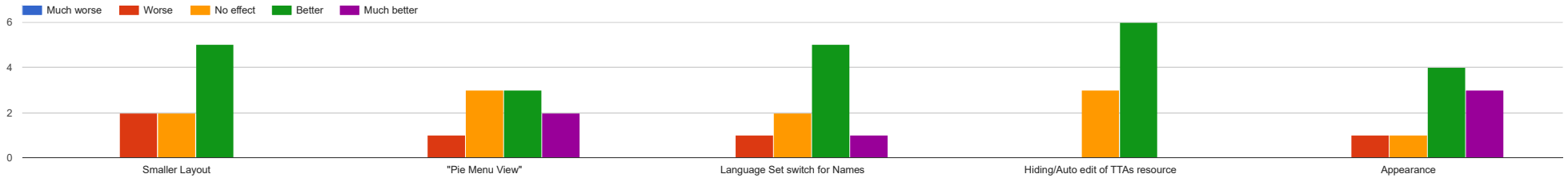
Which tool looks easiest to use? (9 responses)

Which tool looks most powerful? (9 responses)

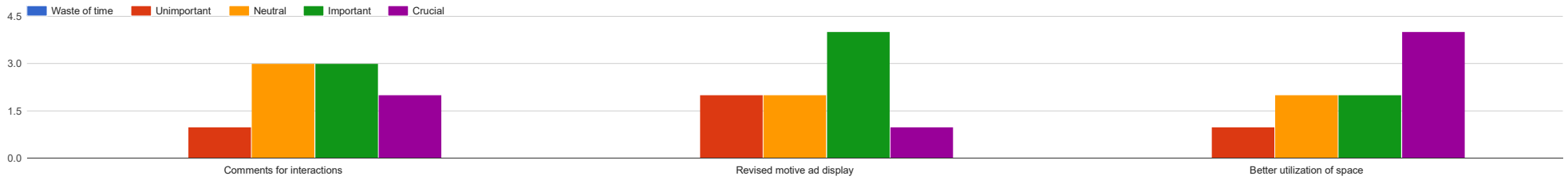
Preview

Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

Rate how you think these features affect the usability of Volcanic in comparison:



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (4 responses)

Preview

There's something about the layout of the Codex that looks better compared to IDE. It's the little details, like having the Pie Menu strings together with the Action and Check trees. The comments section. And the less sense of clutter compared to IDE.

Set the focus to the new line when an interaction is added.

A mass interaction renamer for multiple objects with the same interactions (i.e chairs).

Visual pie menu sample to see how it will look to users ingame

(Optional) What did you like most about the Tree Table editor? (5 responses)

Connecting interactions in Table is very easy

It's very easy to use. Simple, straightforward, makes adding interactions a breeze. If only I knew earlier. Ha!

I liked the simplicity of adding a sub-menu just by using a / in the name.

It's quick and easy to change an interaction name.

Naming interactions is done in this menu rather than the string editors

(Optional) What problems did you encounter? (3 responses)

My primary issue was with Chunk IDs again. Adding a Semi-Global interaction seems to be a BAD thing because it breaks the program (mostly makes an error pop up, nothing biggy, doesn't crash it, just returns an error). But once i figured out it had something to do with IDs then the issue was gone.

none

For getting done what I needed to not much. I just tried one configuration and if it worked in game I'd leave it but if not I would try something else

BHAV Editor

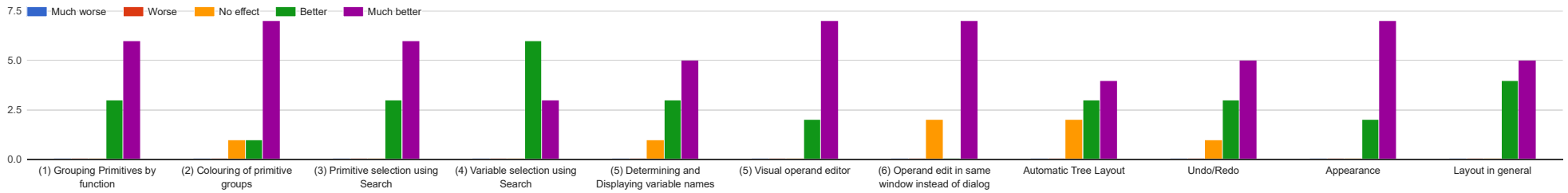
(9 responses)

Which tool looks easiest to use?

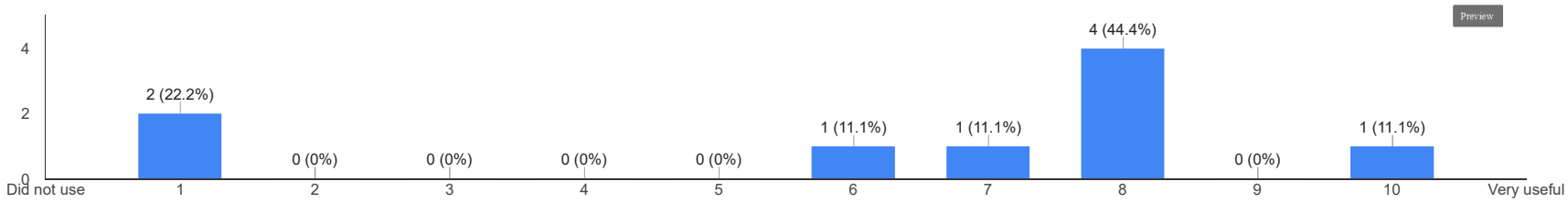
Which tool looks most powerful? (9 responses)

Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

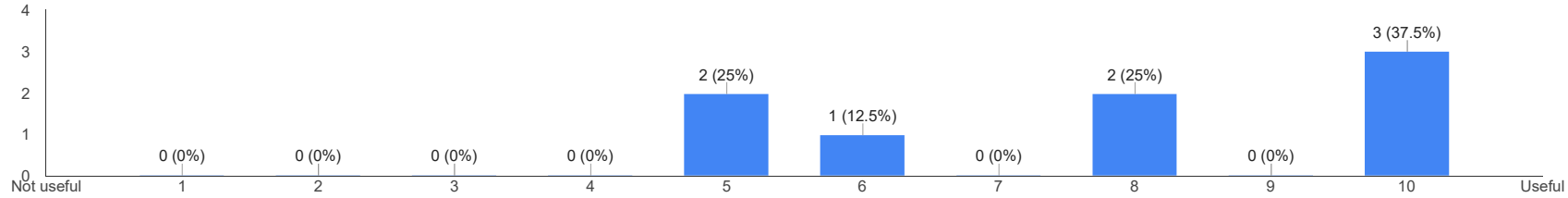
Rate how you think these features affect the usability of Volcanic in comparison (see screenshots for numbered references:



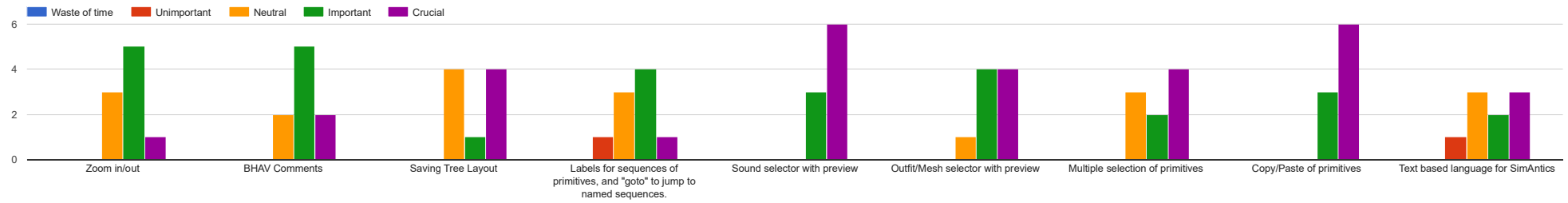
How useful was the "Breakpoint" functionality in Volcanic for debugging trees? (9 responses)



(Optional) When in "Tracer" mode, how useful do you think being able to edit variables on the fly was? (8 responses)



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (2 responses)

Copy and pasting of primitives from the same or other windows.

More primitives being able to be used e.g thought bubble primitive. Documentation.

(Optional) What did you like most about the BHAV Editor? (4 responses)

Compared to Edith and Codex, those two look like a mess. If I had to pick ONE thing that IDE excels, it's this. The way it's created is a very easy to understand, for dummies with color coordination, smooth and eye appealing aesthethics and easy access to everything. When I first saw the existence of BHAV editor I was like "omg, I can ACTUALLY use this" instead of being "wut?" with the other two. It's the simplicity that works in favor in IDE.

The colour coding of primitives makes it easy to follow at a glance, the operand editor always being visible, the search feature to find subroutines.

The colouring and modern design allow you to quickly identify the structure of the interaction.

Feels very common sense driven. I really like it because it feels like anyone who has an understanding of words and the sequence that you want events to take place in with whatever object your changing, it feels like you could make anything you wanted really

(Optional) What problems did you encounter? (2 responses)

Preview

Duplicating previously existing trees is quite a pain and I think they are a lot more complex to do it than meets the eye. But with proper documentation anything will be possible.

Big trees can get cluttered quick which makes them harder to read.

Graphic Editor

Which tool looks easiest to use? (9 responses)

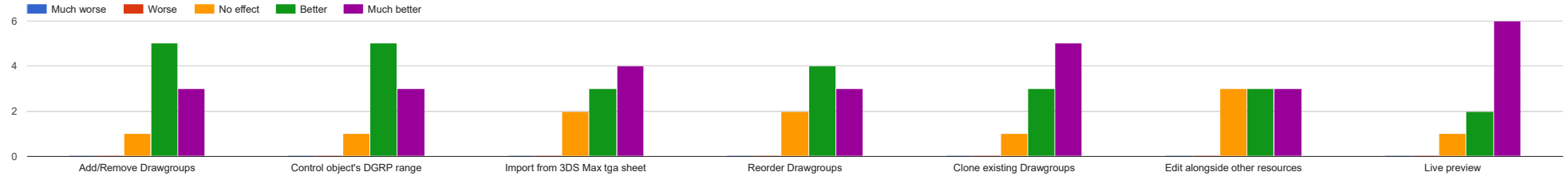
Which tool looks most powerful? (9 responses)

Which tool do you think would be most effective at creating objects for a familiarised user? (9 responses)

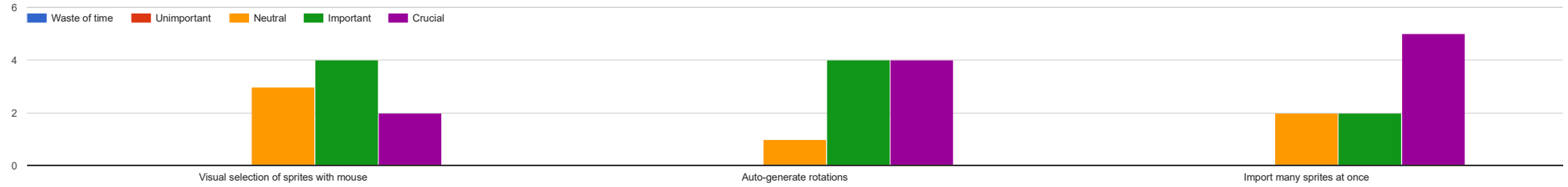
Do you think the presence of the Volcanic DGRP editor detracts from the simplicity of changing sprites? (9 responses)

Preview

Rate how you think these features affect the usability of Volcanic in comparison:



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (5 responses)

Not sure if at all possible, but I was attempting to create a multi tile object, and couldn't. So I had to stick with one tile. Altough I'm not sure "how" one makes a multi-tile object, it doesn't appear like the ability exists within the Appearance tab. So a way to create multi tile objects probably would work in favor of this.

Automatic drawgroups

Generating all rotations for draw groups, auto-creation of alpha images

Autogenerating the depth and alpha sprites

The ability to only import and export colour maps for recolours. Also auto-generated depth and alpha? I'm not sure how hard that would be but it'd make life 100x easier for complicated/animated sprites.

(Optional) What did you like most about the Graphic Editor? (6 responses)

Transofrmeifmsgser is the only program I have previously used (so I can could make buyable fire.. don't judge! Saved me money on that genie lamp). ANYWAY... the thing that the Graphic Editor has a leg up from Trans... the other proram, is that all the buttons are available within a single click, they're all there, no extra dialogs. So that's a huge adavntage over... the other program.

Live preview of object animations

Auto-creation of all zoom levels upon import.

The easy navigation and preview

The live preview helped you understand what the object would look like in game and allowed for quick and easy editing.

Preview

Feels very automatic how all you have to do is pick file folders to import or export to and from respectively.

(Optional) What problems did you encounter? (4 responses)

Multi-tile object editing seems to be a mystery to me right now. It may just be misinformation at this point. Future will tell.

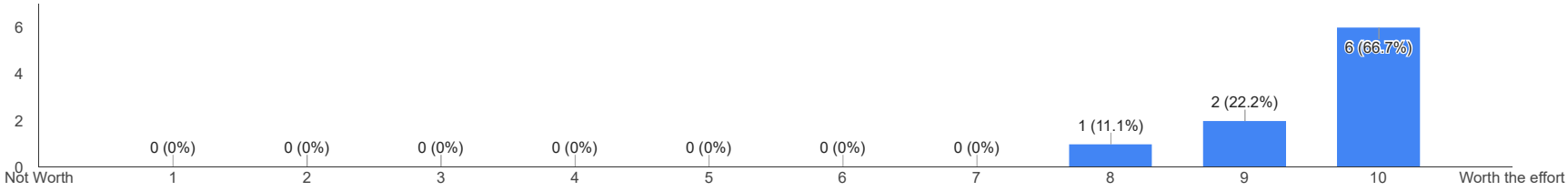
The wrong image size for a sprite was able to be imported, maybe a check could be added to inform less experienced users.

The steps in order to complete a simple recolour can be quite tedious.

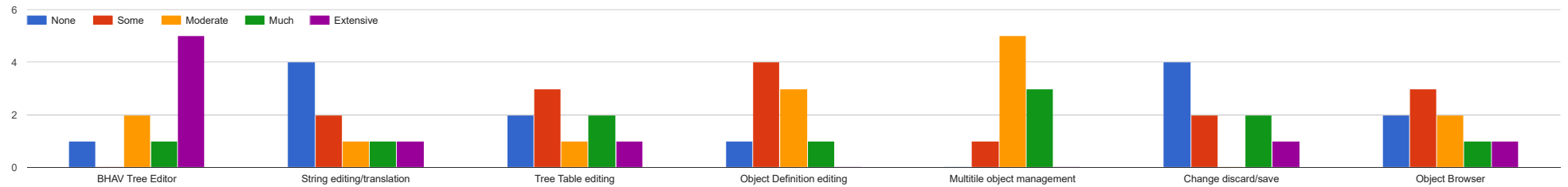
For some reason it wouldn't import my images. I made sure that names sizes and all that matched up. Perhaps a error window with a message stating the error and possible fixes?

Overview

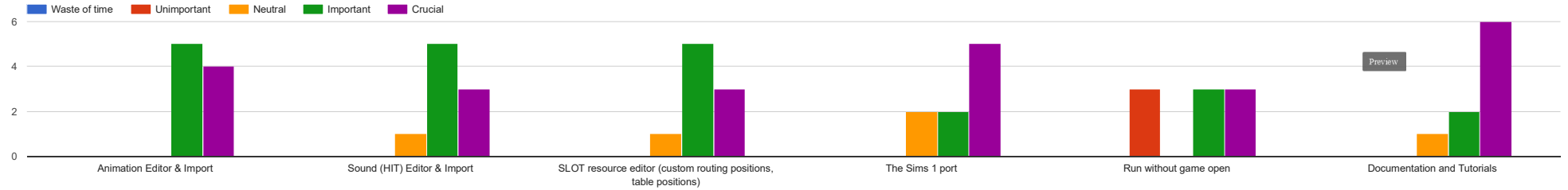
Overall, do you think the development of the tool was worth it? (9 responses)



How much documentation (explaining how to use it) do you think each of these areas requires?



How important do you think the development of these future features is?



(Optional) Suggest any future features you would like to see. (1 response)

The more doors you open the better. Anything that changes what the objects can do, is what will help the development. Imagine being able to create Weather control objects and other stuff like that.

(Optional) Any thoughts, feelings? Please share comments about anything on your mind! (6 responses)

I liked very much to experience the Volcanic , both for fun and learning .. but .. All Forms I say Thank you to restore this game , and make this tool.. you are best

The thing in general is this. Sure, Codex may be slightly better for Strings. Transformigier may seem to have better options. But at the end of the day. Volcanic IDE has about 4? Like FOUR programs in a single software. Which is why it's a MUCH better alternative. Collectively the simplicity of the program wins over the intimidation of the others. So if I had to pick "which program excels the best". I'd pick Volcanic.

- [Removed]

Great work with the IDE, a really useful tool for game modding

Excellent job on creating Volcanic, it is definitely the most well rounded and extensive set of modding tools ever created for The Sims. I'm looking forward to seeing what the future brings for the tools and the game itself! - [Removed]

It's a shame original TS1 never got a tool this great. It honestly helps content creation so much and allows people who may not have a huge programming knowledge to create new and interesting content for the game. Good job.

Thank you very much for taking the time to go above and beyond! You're calling me the best? Hell, I think I should be calling you the best :P

Thank you!

Bibliography

[1] Mobygames. *The Sims*. URL: https://www.mobygames.com/game/sims.

[2] Metacritic. *The Sims for PC Reviews*. URL: http://www.metacritic.com/game/pc/the-sims.

[3] Trey Walker. *The Sims overtakes Myst*. Mar. 2002. URL: http://www.gamespot.com/pc/strategy/simslivinlarge/news_2857556.html.

[4] Daniel Terdiman. *'EA Land' closing just weeks after debut*. Apr. 2008. URL: http://www.cnet.com/news/ea-land-closing-just-weeks-after-debut/.

[5] Rhys Simpson. *About — FreeSO*. Mar. 2016. URL: http://freeso.org/about/.

[6] PC Gamer Magazine. *Slice City: The Sims-Sims*. Apr. 2004. URL: http://www.simslice.com/awards/pcgameruk.htm.

[7] *simlogical - The Sims 1 Downloads*. URL: http://www.simlogical.com/sl/downloadpages/downloads.htm.

[8] BBC News. *Virtual people get their own games*. Feb. 2004. URL: http://news.bbc.co.uk/1/hi/technology/3495285.stm.

[9] Zophar. *IPS Format - ZeroSoft*. 2002. URL: http://zerosoft.zophar.net/ips.php.

[10] Andrew D'Addesio. *Edith cracked — Niotso*. Dec. 2012. URL: http://niotso.org/2012/12/23/edith-cracked/.

[11] Kenneth D.Forbus. *Some notes on programming objects in The Sims*. May 2001. URL: http://www.qrg.northwestern.edu/papers/Files/Programming_Objects_in_The_Sims.pdf.

[12] Don Hopkins. *The Sims, Pie Menus, Edith Editing, and Simantics Visual Programming Demo (reupload)*. URL: https://www.youtube.com/watch?v=-exdu4ETscs.

[13] Lifelong Kindergarten Group. *About Scratch*. URL: https://scratch.mit.edu/about/.