GAM150S14-D

Spring 2014

Neko Means Cat

|  |  |
| --- | --- |
| Tai Der Hui - Producer | Adam Rezich - Technical Director |
| Eduardo Gorinstein - Lead Designer | Samuel Valdez - Product Manager |

Game School Simulator 2015

Technical Design Document

Table of Contents

[Overview 2](#_Toc379820334)

[Neko Engine 2](#_Toc379820335)

[Game 2](#_Toc379820336)

[Spaces 2](#_Toc379820337)

[Entities 2](#_Toc379820338)

[Components 3](#_Toc379820339)

[Archetypes 3](#_Toc379820340)

[Game School Simulator 2015 3](#_Toc379820341)

[Graphics Implementation 4](#_Toc379820342)

[Coding Methods 5](#_Toc379820343)

[Project Layout 5](#_Toc379820344)

[Assets Directory Layout 6](#_Toc379820345)

[Source Control 6](#_Toc379820346)

[Style Guide 6](#_Toc379820347)

[Debugging 7](#_Toc379820348)

[Tools 8](#_Toc379820349)

[Visual Studio 2012 8](#_Toc379820350)

[NekoPak 8](#_Toc379820351)

[SourceTree 8](#_Toc379820352)

[Redmine 8](#_Toc379820353)

[Technical Risks 9](#_Toc379820354)

[Appendix A: Interface Flow 10](#_Toc379820355)

[Appendix B: Art Requirements 11](#_Toc379820356)

[Appendix C: Audio Requirements 12](#_Toc379820357)

# Overview

*Game School Simulator 2015* is a graphical interactive game for Windows, written in C, using OpenGL to render graphics. It uses Alpha Engine as a wrapper for OpenGL, and its own custom engine, Neko Engine, to handle everything else.

## Neko Engine

In order to make implementing design ideas as simple as possible for designers, as well as to create a unified design philosophy, Neko Engine, the game engine for *Game School Simulator 2015*, uses a component-based architecture. Game space is abstracted in terms of *games*, *spaces*, *entities*, and *components*.

### Game

The *game* is the global structure that contains pointers to all of the game systems, such as graphics, sound, and framerate management. It also contains dictionaries of each type of asset (sprites, sounds, textures, and text files) that have been loaded into memory. The game structure also handles the game loop and window events.

The game structure also contains a linked list of *spaces*.

### Spaces

*Spaces* are abstract collections of game objects. Spaces are a convenient way of logically separating game objects. The primary benefit of having multiple spaces in a game is to handle the sorting of game objects’ sprites; spaces further down the game’s list will render later, and therefore be “on top of” spaces rendered before them. Conversely, spaces are updated in reverse order. This allows for behavior such as preventing behavior that usually happens when the user clicks on the background from occurring if the user clicks on a button in the foreground.

Spaces are also capable of having independent frame rates and time scales. This allows for some spaces to be “paused” (set time scale to zero) while leaving other spaces as they are. This can be used, as an example, to pause the main gameplay space while allowing for a separate menu space to remain unpaused, so the entities in the menu space can continue handling input as normal while the game space freezes in place.

In the context of *Game School Simulator 2015* specifically, this also provides the ability to have a separate space exclusively to process the simulation in. This way, the entities can exist in the simulation space as abstract representations of data, interacting with each other, instead of on-screen game objects. The time scale of the simulation space can also be modulated at will, allowing the designers to fine-tune the optimal rate at which the simulation occurs, or even to allow players to adjust the simulation rate themselves.

### Entities

*Entities* are the all-purpose game object structure in Neko Engine. Each entity “lives” in a space, and spaces contain a list of entities. Entities are little more than a container for components, optionally with a uniquely-identifiable name. Entities can be *attached* to one another, allowing them to move and transform as a single unit, or otherwise act connected in some manner as described by component designers.

### Components

*Components* are the core of Neko Engine, and the basis for how everything in it works. As previously described, entities contain no game logic of their own, and are entirely incapable of doing anything outside of abstractly attaching themselves to each other. Components contain function pointers that are invoked at specific points in the game engine. These are called *events*. Components can also access the entity they are connected to, their “owner.” This is important, because from here, they can access the space the owner exists in, the game that space exists in, the parent entity of the owner, the children of the owner, and, most importantly, other components connected to the owner. This allows for components to intercommunicate, to pass information back and forth as each component handles its own discreet behavior. Components also have the ability to have *dependencies*; some components require the existence of other components in order to function properly, and a system exists to prevent designers from accidentally adding components to entities without having previously connected any prerequisite components.

### Archetypes

*Archetypes* allow designers the ability to define “recipes” for the creation of entities, by defining which components should be attached and how they should be initially configured. These archetypes can later be created with ease.

## Game School Simulator 2015

# Graphics Implementation

The graphics of the game are entirely sprite-based, loading images from PNG files and displaying them on the screen using the Sprite component. Neko Engine also has support for larger “texture maps” that can contain several images in a single file, using Sprite Definition files (.SPR) to define the texture name, width, height, and UV coordinates of a given sprite.

# Coding Methods

## Project Layout

The project directory for *Game School Simulator 2015* is laid out as follows:

|  |  |
| --- | --- |
| / | Root directory. Contains Visual Studio solution file, README.md, and any scripts that might need to be frequently run. |
| AlphaEngine/ | Contains Alpha Engine header files. |
| bin/ | Visual Studio project is set to output game binaries here. |
| etc/ | Any assorted files related to the project such as this document are stored here. |
| fmod/ | Contains FMOD header files. |
| Game School Simulator 2015/ | Project folder for the game itself. Contains all C source and header files, as well as Visual Studio project files. |
| data/ | Contains all assets for the game. (See “Assets Directory Layout” on page 4 for more information.) |
| lib/ | Contains all dynamic library files to be included in the build of the game (copied over automatically to output directory), as well as the static object library files for the libraries used in the project. NekoEngine outputs to this directory. |
| NekoEngine/ | Project folder for the game engine. Contains all C source and header files, as well as Visual Studio project files. |
| NekoPak/ | Project folder for NekoPak[[1]](#footnote-1). Contains all C# source files and Visual Studio project files. |
| tmp/ | Temporary directory used by Visual Studio to build the various projects. |
| tools/ | Contains binaries of all custom tools used in development. Output directory for NekoPak. |

## Assets Directory Layout

Game assets for *Game School Simulator 2015* are stored in a folder called “data” in the “Game School Simulator 2015” subdirectory of the project root directory. Assets are then further organized into subdirectories by type, as follows:

|  |  |
| --- | --- |
| bgm | Background music files (.MP3 or .WAV) |
| cfg | User configuration files (.CFG) |
| sfx | Sound effect files (.MP3 or .WAV) |
| spr | Sprite image files (.PNG) and sprite definition files[[2]](#footnote-2) (.SPR) |
| tex | Sprite map texture files (.PNG) |
| txt | Text files (.TXT) |

These directories may contain any number of subdirectories as the designers see fit; the asset pipeline will load them in automatically[[3]](#footnote-3). When a debug build of the project is compiled, the entire “data” directory is copied into its output folder. For release builds of the project, the assets are instead packed into a single “data.pak” file using NekoPak. (See page 7 for more information on NekoPak and how it works.)

## Source Control

Source control is implemented using Git. Repository hosting is provided by Bitbucket, hosted in a private repository at <https://bitbucket.org/adamrezich/neko>. Backups are kept on a local server, as well as on DigiPen version control servers at <https://git.digipen.edu/projects/nekomeanscat>. All team members have learned how to efficiently use SourceTree[[4]](#footnote-4)

## Style Guide

Fill me in.

# Debugging

Fill me in.

# Tools

## Visual Studio 2012

The primary tool used in the creation of *Game School Simulator 2015* is Microsoft Visual Studio 2012 Ultimate.

## NekoPak

Few professional games leave their assets exposed in a subdirectory of the game folder, and *Game School Simulator 2015* aims to be no exception. This is the reasoning behind the creation of NekoPak, a custom command-line tool written in C#. NekoPak packs all of the assets together into a single “data.pak” file, using the PAK file format specifications from id Software’s *Quake* and *Quake II*. The PAK file format does not use compression, making it relatively simple to implement in C and C# without the use of complex algorithms. Being a command-line tool, Visual Studio has been configured to automatically run it on the assets folder after a successful build.

Packing assets in this fashion allows for other benefits as well, in addition to making the finished game directory look tidier. As the asset data is loaded into memory to be packed, additional processing can occur at this time. Since Alpha Engine lacks the ability to read height and width information from PNG files as it loads them, NekoPak can make up for this by encoding the height and width data directly into the PAK file. Then, Neko Engine can load this information at runtime, along with the image data itself.

## SourceTree

Neko Means Cat in its entirety has been trained to use SourceTree effectively, as a way to interface with the project’s Git repository.

## Redmine

A locally-hosted instance of Redmine is being used as the project’s issue tracker and project planning software. This allows the producer to assign tasks to the rest of the team members, along with deadlines. Other team members can also use the software to file any bugs they encounter.

# Technical Risks

The primary technical risk inherent to the design of *Game School Simulator 2015* is the potential of memory leaks inherent to the design of the complexity of the Neko Engine, given its relatively untested nature. Nearly every aspect of the game engine involves dynamically-allocated memory that must be kept track of and freed properly, especially with the way students enter and exit the school over time. Care must be taken to ensure that all allocated memory is freed when it is supposed to be, and that no leaks crop up as the result of poor memory management.

Another technical risk of the game is the difficulty involved in connecting the simulation space to the visual spaces in a way that is meaningful and engaging for the player. With the system currently in place, creating the simulation itself is not difficult, and neither is putting objects on the screen. The risk involved is the possibility that connecting the simulated elements of the game (rooms, students, passage of time) to the on-screen, visual space of the game will be more of a challenge than anticipated.

# Appendix A: Interface Flow

Fill me in.

# Appendix B: Art Requirements

Fill me in.

# Appendix C: Audio Requirements

Fill me in.

1. See “NekoPak” on page 7. [↑](#footnote-ref-1)
2. See TODO: FILL IN [↑](#footnote-ref-2)
3. See TODO: FILL IN [↑](#footnote-ref-3)
4. See “SourceTree” on page 7. [↑](#footnote-ref-4)