NTNU
Norwegian University of
Science and Technology

Noxed - A NOX Engine World Editor

Author(s)
Gisle Aune
Tor Strømme

Bachelor in Game Programmering
20 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

18.05.2016

Supervisor(s) Simon McCallum

Sammendrag av Bacheloroppgaven

Tittel:	**Noxed - En Verdensregissør bygget med NOX Engine**
Dato:	18.05.2016
Deltakere:	Gisle Aune Tor Strømme
Veiledere:	Simon McCallum
Oppdragsgiver:	Suttung Digital
Kontaktperson:	Magnus Bjerke Vik, magnus@suttungdigital.com
Nøkkelord:	verdensredigeringsprogram, c++, fliskart, 2D
Antall sider:	50
Antall vedlegg:	3
Tilgjengelighet:	Åpen

Sammendrag:

Noxed er et 2D nivåredigeringsprogram bygd for – og med – NOX Engine. Den utnytter JSON filformatet for aktører og verdner for å konstruere en fil som kan bli forstått av den innebygde innlasteren i spillmotoren. Dette gjør det enklere å bygge nivåer uten den slitsomme prosessenen som er å endre store JSON filer manuellt som bare blir værre som verdnene vokser i størrelse og kompleksitet. Programmet er konfigurerbart av spillutvikleren, som har full kontroll på hvilke aktører som kan plasseres, og hvor vidt deres alternativer kan redigeres av nivådesigneren. I tillegg til dette har Noxeds utvidelser av motoren en metode for å bygge modulære grafiske strukturer ut fra sammenhengssensitive fliser.

Summary of Graduate Project

Title:	**Noxed - A NOX Engine World Editor**
Date:	18.05.2016
Authors:	Gisle Aune Tor Strømme
Supervisor:	Simon McCallum
Employer:	Suttung Digital
Contact Person:	Magnus Bjerke Vik, magnus@suttungdigital.com
Keywords:	world editor, c++, nox, tilemaps, 2D
Pages:	50
Attachments:	3
Availability:	Open

Abstract:

Noxed is a 2D level editor built for – and with – the NOX Engine. It leverages the JSON file format of actors and world files to construct a world file that can be understood by the built-in means of world loading within the NOX Engine. It eases the process of creating levels for the games without the tedium of manual JSON addition or reliance on procedural generation to make game worlds of scale. The editor is configurable by the game developers, and they can manage the actors that can be used and expose options within their components for the level designer. In addition to that, the Noxed extensions to the engine provides a means of constructing modular graphical tile maps out of smart, context-sensitive tile maps.

Preface

In August 2015 we decided to see if Suttung Digital would be our employers on a bachelor project. After meeting with them in November they accepted our proposal of creating a world editor with the NOX Engine.

We would like to thank Simon McCallum for being our project supervisor. A special thank you goes out to our principal Suttung Digital; Magnus Bjerke Vik, Håkon Nordling, Jakob Solheim and Tomas Olsen. They gave us valuable feedback and encouragement that were a big help during the development of our editor.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Goals and Framework

1.1.1 Background

We are two students studying Game Programming at NTNU Gjøvik. Our wish is to create a 2D level editor for a game made with the NOX Engine made by Suttung Digital. Our main focus is to get an editor that can be used to generate levels without any need to change the game.

We want to have support for tile sets and game actors.

1.1.2 Target Audience

This project is about creating a tool for level designers, which is meant to interface with content created by the game developers and graphics designers.

This document is mostly targeted towards developers who are looking to use the Noxed editor and extensions to the NOX Engine in their project, but there are guides and information pertinent to usage that do not involve programming.

There are guides offered in chapter 4 that goes over the process for other team members to set up the editor for level designers' use.

- Add editor-actor: 4.1.3
- Add tile set: 4.2.3
- Add tile type: 4.2.5
- Add UI element group: 4.3.3

1.1.3 Project Goals

Effect Goals

- Get a deeper understanding of how games and level editors interact.
- Learn more about tool development. Creating tools that are practical and usable.

Result Goals

- The editor should be usable and complete.
- The basics aspects of the editor should be understandable and intuitive.
- The levels created should work on the game without changing or modifying the game.
- The games and editor should work in both Windows and Linux.

1.1.4 Project Framework

We will use free development tools, either as in available to students free of charge, or open source. If we need additional libraries, we will opt for free options with an open license. There are no costs associated with the project.

NOX Engine

NOX Engine is the game engine we will use to build our editor and prototype games. It is an open-source 2D engine that runs on Linux, Windows and OS X. It uses SDL2 to create windows, and OpenGL to render on screen. Actors and levels are stored as JSON files. Our plan is to use our editor to read in dummy versions of actors and place them in a running game world. When the world is complete we will serialize the world to a JSON file using the real version of the actors.

1.2 Scope

1.2.1 Subject area

We would like to create an editor that is easy to use, with objects that are highly modifiable. We will focus on the following areas:

- The games using this editor shall be able to expose any number of game objects by using special assets.
- The editor shall save world information in the format the engine already uses.
- The game can use a core set of objects and features so it can understand the Noxed actors without including the entire level editor.

1.2.2 Limits and Requirements

The game(s) and editor should run in Windows 10 or newer and Ubuntu 14 LTS. The programs should start on integrated graphic cards, but need only run smoothly on newer dedicated graphic cards. This requirement does not account for level designs with a high object density.

1.2.3 Assignment

We will make at least one game and a level editor to said game. If time permits we will make multiple games and make the editor work on all of them.

1.3 Project Organization

1.3.1 Suttung Digital

Suttung Digital [1] is a local game and software development studio founded by students who attended Gjøvik University College (now Norwegian University of Science and Technology - Gjøvik). They are our principals on this project.

1.3.2 Responsibilities and roles

Team Leader: Tor Strømme

No other roles defined seeing as we are only two persons.

1.3.3 Routines and Rules

- Any decision made for the group should be discussed internally first. If no agreement can be reached we will first discuss with an external (Magnus from Suttung or Simon from NTNU Gjøvik). If no agreement is reached, the group leader will make a decision.
- If a member of the group has not performed his assigned tasks in the agreed upon time it is up to the other member to give a verbal warning. If the behavior is re-

peated a meeting will be called with Simon.
- Any unforeseen costs will be split evenly among the members.
- Any member has the right to sign and authorize on behalf of the group.
- When work hours are concerned our main objective each week will be to complete assigned tasks. However we expect each member to spend at least 20 hours each week even if the task is easy or completed early.

1.4 Planning and Reports

1.4.1 Work flow

We will follow the Scrum model as much as is sensible with two persons. Mainly we will have four major parts: Design, develop game, develop editor and completion. The game and the editor will be developed in tandem. We allocated 9 weeks starting week 6 to develop our software. The remaining time will be used for bug fixes, polishing and report writing.

We will start with weekly sprints working together in the same room, though this is subject to change as the project matures. If we do not physically meet on a work day we will have the daily scrum meeting on Skype instead. Every Monday at 13.00 we will hold the Scrim Review and Retrospective for the completed sprint followed by the Planning Meeting for the next sprint. Any member can call meetings and all members are expected to update the Backlog.

1.4.2 External Meetings

Meet with Suttung Digital once a week, Mondays 1400. Simon will be at the Mustad rooms we are using at Wednesdays so we will have meetings with him there approximately every other week.

1.5 Quality Assurance

1.5.1 Documentation

All commits to Bitbucket should be complete with useful comments(see 1.5.3). Classes, functions and resources should be commented in a fashion that we can generate documentation using Doxygen or similar software.

1.5.2 Code Style and Quality

We will attempt to use the c++ core guidelines.

1.5.3 Configuration Management

Issues and tasks will be tracked using Jira. We chose Jira since it is popular and we figure that it is easier to get help if lots of people use it.

Git will be used for source control. Any errors with integration will be reported when discovered. The author is responsible to ensure that their additions to the code will integrate into the project, and should be available to troubleshoot any issues during integration.

Every new commit to the master branch will be preceded by a code review with the other group member. This is to ensure that the latest version always is working and satisfies all standards.

When we get a working copy of the game and editor the newest versions will be made available on the NTNU bachelor project web site.

1.5.4 Risk Analysis

Event	Probability	Impact Assessment
Sickness or disease	Medium	Depends on duration and severity
Absence and procrastination	Medium	High
Hardware issues	Low	Low
Technical issues with game engine	High	Low
Downtime on web services	Low	Low
Lack of Communication	Medium	High
Loss of Group Member	Low	High

Table 1: Project Risks

Sickness and Disease Notify the other group member. Make sure any work you have done is documented and available.

Absence and Procrastination Communicate what is happening on the group meetings. Motivate each other.

Hardware Issues Make sure you save and commit your work regularly. We both have backup computers.

1.6 Work Planning

1.6.1 Main Parts

We have some flexibility in the planning phase since we are using an agile development model. We have agreed on five major milestones we want to reach at certain times. What tasks need to be completed for each milestone is yet to be seen.

1. **Project Plan Delivery**
   28/01-16
2. **Prototype of Game and Editor**
   22/02-16
   - A basic user interface for the editor and a game with a player that can move around.
3. **Main development complete**
   10/04-16
   - The game and the editor is finished and complete. Only bug fixes and polish remains.
4. **Final Sprint**
   09/05-16
   - Any and all development from here on out will be if the code does not compile and run on the target systems. Code unrelated to that has to be committed to be merged or discarded by this date.

5. **Report Delivery**
   18/05-16

1.6.2 Assignments

We have planned on using 9 weeks for the main development cycle, but leave the last 5 for the written report, bug fixing, polishing and catching up if a feature took more time than first anticipated. We have five tasks in our main cycle and one task after the development is done.

Main Development Cycle

1. Blank sheet to work from. **1 week.**
2. Basic editor with placement of assets. Simple prototype of a game. **2 weeks.**
3. Tile sets in both game and Editor. **2 weeks.**
4. Interface in editor with object properties. **2 weeks.**
5. Complete game(s). **2 weeks.**

After Development

1. Report writing, bug fixes. **Remaining time.**

This is a list of features we would like to implement, in descending order of importance not order of implementation. If time is an issue we will discard from the bottom first.

- **Editor:** Entity placement.
- **Editor:** Entity properties.
- **All:** Tiled graphics for 2D level geometry
- **Game:** One example game with simple fighting, enemies, and movement.
- **Editor:** Tile set placement.
- **Editor:** Different brushes.
- **All:** Automatic adjacency-adapting tiles for faster tile map production.
- **All:** Meta-data for tiles that can be used by the game and editor.
- **All:** Integration of editor as submodule in projects.
- **All:** A separation of editor and core objects needed both by editor and game.
- **Game:** Another example game with basic tower defense mechanics where a player can edit the map between attempts
- **Game:** Sound effects.
- **Game:** Music.

1.6.3 Gantt Chart

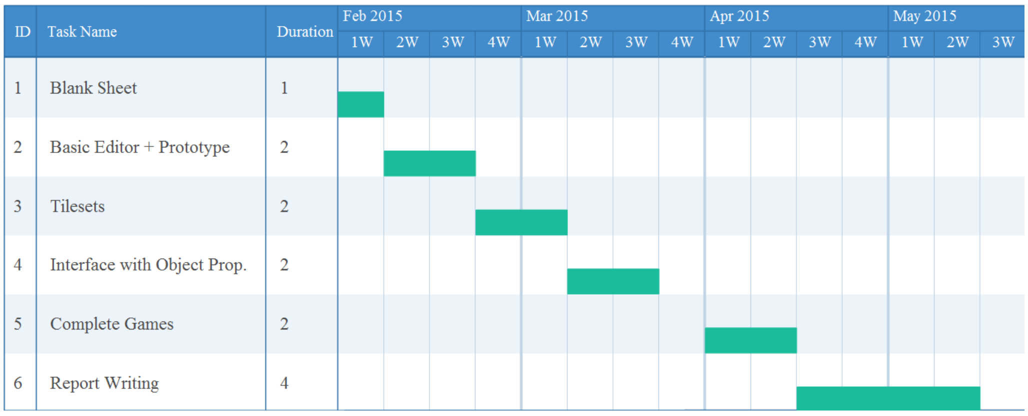


Figure 1: Gantt chart.

2 Requirements

2.1 Functional requirements

We are developing a level editor to provide a visual means of constructing worlds for Suttung Digital's NOX Engine. It shall allow placement of objects and tiles defined by the game developers for the editor's target game, and export them into the JSON format that the game can read.

To do so, Noxed needs to have a way to load the necessary assets and present them to the level designer with the user interface. The specific assets it needs are:

- The cursor controller placed in the editor world.
- All JSON files for tile set definitions under `tileset/`
- All actor definitions referred to in the cursor controller's definition.

There also needs to be a cursor that the level designer can use to add, select and remove content to the level. It should be capable of operating on tile maps and actors alike, but not at the same time. Modes set by keyboard input or UI input shall control the cursor's behavior to decide whether it is acting upon a tile map or the actor space, how it does so and which specific tile or actor it places.

The editor should allow some minor adjustments for the location of UI elements, and key rebinding. Those features does not need to be configurable in the program itself, configuration files will suffice.

2.1.1 Use case diagram

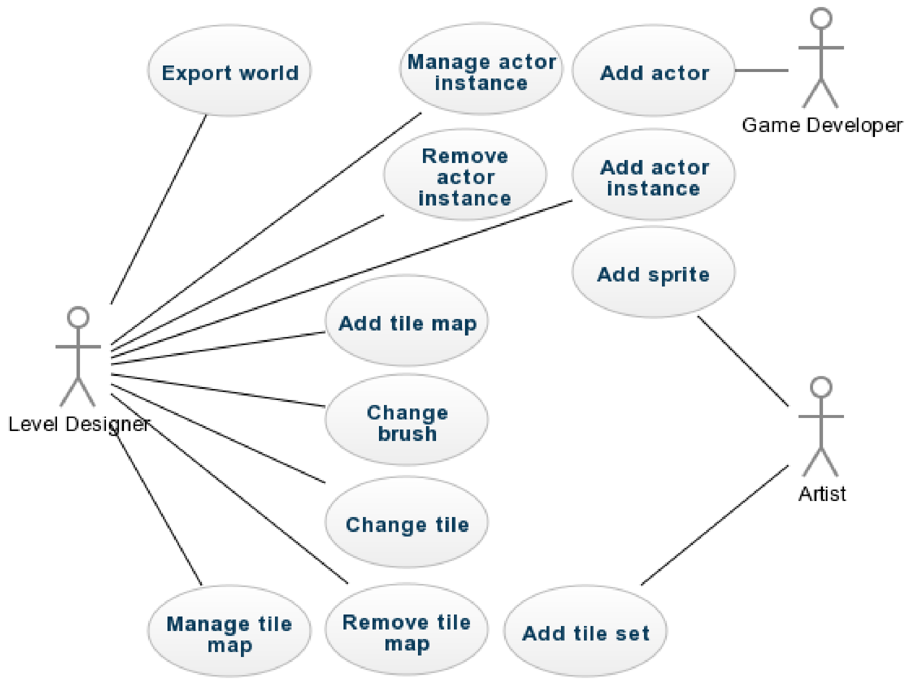


Figure 2: Use case diagram

2.1.2 Use cases

Use Case	Change tile
Actor	Level designer
Goal	Change the tile at a specific position, or positions on a tile map.
Normal Flow	1. Set the editor mode to activate the tile spawner. 2. Select the tile map from the tile map list. 3. Move the cursor to the correct location. 4. Left click to place the tile.
Variations	When removing a tile, use right click instead of left click.

Table 2: Use Case: Change tile

Use Case	Change brush
Actor	Level designer
Goal	Change the size and/or shape of the brush.
Normal Flow	1. Set the editor mode to activate the tile spawner. 2. Adjust the brush size slider or use the configured keyboard shortcut. 3. Adjust the brush shape slider.
Variations	None

Table 3: Use Case: Change brush

Use Case	Add actor instance
Actor	Level designer
Goal	Add an actor at a position in the world.
Normal Flow	1. Set the editor mode to activate the tile spawner. 2. Select the actor by using the slider or the configured to change selection. 3. Move the cursor to the correct location. 4. Left click to place the actor.
Variations	None

Table 4: Use Case: Add actor instance

Use Case	Remove actor instance
Actor	Level designer
Goal	Remove an actor at a position in the world.
Normal Flow	1. Set the editor mode to activate the tile spawner. 2. Select the actor by using the slider or the configured to change selection. 3. Move the cursor to the actor's location. 4. Right click to remove the actor.
Variations	None

Table 5: Use Case: Remove actor instance

Use Case	Manage actor instance
Actor	Level designer
Goal	Change an aspect of an existing actor instance in the world.
Normal Flow	1. Set the editor mode to activate the tile spawner. 2. Select the actor by using the slider or the configured to change selection. 3. Move the cursor to the actor's location. 4. Left click on an existing actor to select it. 5. Change the relevant GUI sliders and inputs.
Variations	If you want to manage the actor that was just spawned, then.

Table 6: Use Case: Manage actor instance

Use Case	Add tile map
Actor	Level designer
Goal	Add a tile map to the world.
Normal Flow	1. Select the tile set from the tile set list. 2. Click the button to add a tile map. 3. Observe the change in the tile map list.
Variations	None

Table 7: Use Case: Add tile map

Use Case	Manage tile map
Actor	Level designer
Goal	Change one or more properties of a tile map in the world.
Normal Flow	1. Select the tile map from the tile map list. 2. Change the relevant properties.
Variations	None

Table 8: Use Case: Manage tile map

Use Case	Remove tile map
Actor	Level designer
Goal	Remove a tile map to the world.
Normal Flow	1. Select the tile map from the tile map list. 2. Make very sure it is the correct one you have selected. 3. Click the remove button.
Variations	None

Table 9: Use Case: Remove tile map

Use Case	Export world
Actor	Level designer
Goal	Remove a tile map to the world.
Normal Flow	1. Finish up the level editing. 2. Press Left CTRL + S. 3. The data is now saved to `assets/world/initial.json`.
Variations	The key combination mentioned in step 2 may have been remapped in the controls.json. The file name of the exported world may also have been configured to be different by changing the component definition for the `Noxed::WorldSaver` component.

Table 10: Use Case: Export world

Use Case	Add actor
Actor	Game developer
Goal	Add an actor to be used by the editor.
Normal Flow	1. Add a new actor with a unique name. 2. Add a physics, transform and a visual component to the actor. 3. Add a `Noxed::EditorBlueprint` component that has `"extends"` set to the correct game actor. 4. Add components (if any) to the blueprint.
Variations	When doing this for the first time, I advise reading the guide in section 4.1.3 that explains the steps in greater detail.

Table 11: Use Case: Add actor

Use Case	Add sprite
Actor	Artist
Goal	Add a sprite to the editor actor as a visual component
Normal Flow	1. Add image to sprite sheet. 2. Add or locate Sprite component in actor you wish to add it to. 3. In the sprite field, write the file name of your sprite.
Variations	The project can use a different sprite component that is not referred to as "Sprite", and it would need to be set up differently if it is animated.

Table 12: Use Case: Add sprite

Use Case	Add tile set
Actor	Artist
Goal	Add a tile set to be used by the editor.
Normal Flow	1. Add the source image to the image atlas. 2. Note the tile size, count and separation. 3. Enter that data into a new tile set definition. 4. Add appropriate smart tiles to the tile set definition. 5. Save it.
Variations	When doing this for the first time, I advise reading the guide in section 4.2.3 that explains the file format and steps in detail.

Table 13: Use Case: Add tile set

2.2 System Requirements

The editor is made to support both Windows 10 and Ubuntu 14.04 LTS. That is not to say that it wouldn't work on other versions of Windows and distributions of Linux, as it likely will, but the requirement is that it work on those platforms. None of us have a device running Apple's OS X, which is the reason for its omission here.

The editor needs to support graphics cards and drivers which support OpenGL 3.0 and up. Modern dedicated graphics card generally do meet this requirement, while Linux drivers for intel's integrated graphics do not. There should be no guarantee of full functionality when using a system that does not meet this requirement.

The editor will have different requirements than the games that it edits levels for. In some regards it will only display static representations of dynamic actors, but on the other hand the smart tile map may require more processing than the game tile map if the option to export only the result is selected.

3 Process

All planning will be done in January. We expect to get ideas for minor improvements and additions underway, but all major features of the editor should be agreed on and be on a prioritized list when we start our programming.

3.1 Workflow

As mentioned in our introduction we had 5 major milestones that we wanted to complete. For each of these the work was organized into *issues* and *sprints*, with a *project backlog*. The following is our definition of the terms as we used them.

Issue: Any improvement, bug fix or addition to the software code itself or its assets.

Sprint: One week starting on Tuesday and ending on Monday.

backlog: All issues not currently added to a sprint or resolved.

Any work to be done on the project needs to belong to an issue. An issue needs to be described and receive an estimation of time needed to complete it. If the time assessment is over thirteen hours we will attempt to split the issue into several smaller issues. When an issue is created it will be added to the project backlog.

Every Monday we will have a meeting where we will decide on which issues should be added to the coming sprint. If we fail to complete any issues we will first attempt to fit it in the next sprint, if that is not possible it will be added to the backlog or discarded.

An issue will have 1 to 13 story points corresponding to the number of work hours estimated for the issue. Story points are used to keep the workload for each sprint appropriate.

3.1.1 Workflow Interruptions

Bugs

With the NOX Engine being a work in progress we expect minor bugs. We also expect to deal with them quite easily since the engine is open source. If we find a bug in their engine we will take the following steps in order:

- Ask Suttung Digital if there is a fix not yet merged into the NOX Engine master branch.
- Fix the bug ourselves and send a pull request.

Missing Features

If there is a feature missing we have a different approach:

- Ask suttung about an alternative way of adding the desired functionality.
- Consider dropping the functionality that needs this feature.
- Fork the NOX-engine and add the feature ourselves.

Forking would really only be considered if it was critical to finishing our project. Adding features to an engine runs the risk of quickly running up unnecessary time and may prove to be too complicated.

3.2 Project tools

3.2.1 Git

https://git-scm.com/

> "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency." [2]

Having very little experience with other types of source control the choice to use Git was easy. Our project is hosted on Bitbucket and contains two repositories, one for the code and one for our Thesis. In addition we had both our interface library and the NOX-engine added as submodules. Submodules are a way to add existing repositories as sub directories to your repository without them interfering with each other.

Working with git we decided to give each issue it's own branch on Bitbucket to keep the master branch clean in the sense that it would always compile and run. When merging completed work into the master branch there should be a short code review. This is to ensure that the code was good and that the code runs on both Windows and Linux.

3.2.2 Jira

https://www.atlassian.com/software/jira[3]
Jira is a tool that is used for planning and tracking progress during the development of software. Although Jira is a very powerful and comprehensive tool, we mainly wanted to use it for issue creation, issue tracking and planning sprints. When using Jira and Bitbucket together you can also use smart commits. Adding keywords to your git commit message will automatically update the status of the issue in Jira's web interface.

3.2.3 Slack

https://slack.com/[4]
Slack is a simple chat program with servers and channels. Slack is server based with open and private channels. Apps are available for Android, iOS, OS X, Linux and Windows in addition to a web interface. Recommended by our principal Suttung Digital.

3.2.4 Toggl

https://www.toggl.com/[5]
Toggl is an advanced stopwatch application with both desktop and web versions. You can track time within a project and with specific tags. There are also advanced report features where you can see time spent per week and what tags or projects you spent the most time working on.

3.3 Programming Tools

3.3.1 CMake

https://cmake.org/[6]
We are doing cross-platform development. CMake is a tool that helps us automate the build process and in our case generate makefiles for Linux, and solutions and projects for Visual Studio on Windows. Suttung Digital has an example template we expect to easily adapt to our project. Using Cmake also cuts down on the setup time for a development environment on a new machine.

3.3.2 Visual Studio

https://www.visualstudio.com/[7]
For development on Windows we will use Visual Studio Community 2015 with the Visual C++ compiler. The IDE was chosen because it is free to use and feature code completion and debugging.

3.3.3 Atom

https://atom.io/[8]
Atom is a highly modifiable text editor we will use for developing on Linux. It is free and offers some auto completion and syntax highlighting.

3.3.4 TeXstudio

http://www.texstudio.org/[?]
We are using latex to render our thesis paper. TeXstudio is an open source integrated writing environment. We chose it because it is free and has several quality of life features like autocomplete, jumping within the document, previewing and syntax highlighting.

4 Technical Details

This chapter will go over the technical aspects of the level editor, and Noxed's extensions of the NOX Engine. First and foremost we will talk about the editor blueprints, tile sets and the graphical user interface, but all components and events are described at the end of this chapter.

4.1 Editor Blueprints

The editor needs to understand the game, what actors can be placed and what settings the level desiner should be able to tinker with. That is where editor blueprints come in. They are a component of an static actor that represents visually how it will appear in the game.

4.1.1 Design

The idea behind this is to place customizable objects, with options and ranges that the developer sets by intention. It does not inherently understand NOX Engine actors components aside from the Transform. Instead, the blueprint contains settings for how an object is exported. Specifically, what actor it extends, whether it's controlled, and the components. The components can base itself on a pre-written JSON object, or the one in the actor representing the game actor in the editor.

When placed or selected, the actor options appear in the UI with sliders, text inputs, checkboxes and so on. These settings will not have an immediate effect on the in-editor appearance of the actor.

4.1.2 General Usage

To start, you want a static actor with a Transform, a visual component (e.g. Sprite) and physics. Physics is used to check for occupancy, which is used for deletions and preventing overlap. If you have the editor assets and game assets in the same folder, you should use a suffix or prefix for the actor names to avoid ambiguity.

In that actor, there should be a Noxed::EditorBlueprint component. Default values for "extend" and "controlled" are unset and false respectively. The last saved actor with an editor blueprint that has "controlled" set to true will be the controlled actor in the game.

Components are added to the blueprint similarly to how they're added to actors, but the layout inside them differs. First you have to choose whether the component definition is based on the editor representation's component, a JSON object, or nothing. The former is useful for editable components like the tile map, as well as component that you want to base on the editor component. "base" is an option if you don't use the editor component, and as its name implies, it will serve as the base for the arguments to write upon.

The arguments inside "args" are applied last, and these are what the GUI will display. They have the following arguments.

- "type": The type
- "value": The default value of the argument
- "min": The minimum value of the argument
- "max": The maximum value of the argument
- "path": An array of field names that are traversed into before the argument is written

The following types are implented. Please note that for numeric values, a minimum and maximum value are required.

- "boolean": Checkbox
- "int": Slider
- "float": Slider
- "string": Text Input
- "vector": Two float number inputs

4.1.3 Guide: Add a player character

Before I begin, the scope of this guide is the JSON side of things, and it assumes you already have the components in order n the game. It also takes for granted that you have copied the cursor object over from the example and changed the values to make it fit with your setup.

This process can be repeated for any actor, however it does not demonstrate basing the constructed component on another. The "E_TileMap" actor included with the editor assets are an example of a rare use case for a serialization of the in-editor component.

Make E_Player.json

You start off by adding the JSON for the editor object. It should be a static representation of the object in game. The required components are the Transform and Physics, for they will be used to have the object take up space in the game world and allow selection and deletion of it. Any visual components like Sprite and Light are optional, but strongly recommended.

The prefix is used to distinguish editor actors from the actual in-game actors in the editor.

However, the editor doesn't know what to do with this object when the level is exported. For that, a Noxed::EditorBlueprint component is required. This is going to be the controlled actor. See the usage section above and the example below to fill in the sliders, fields and checkboxes for the player controller component in your game.

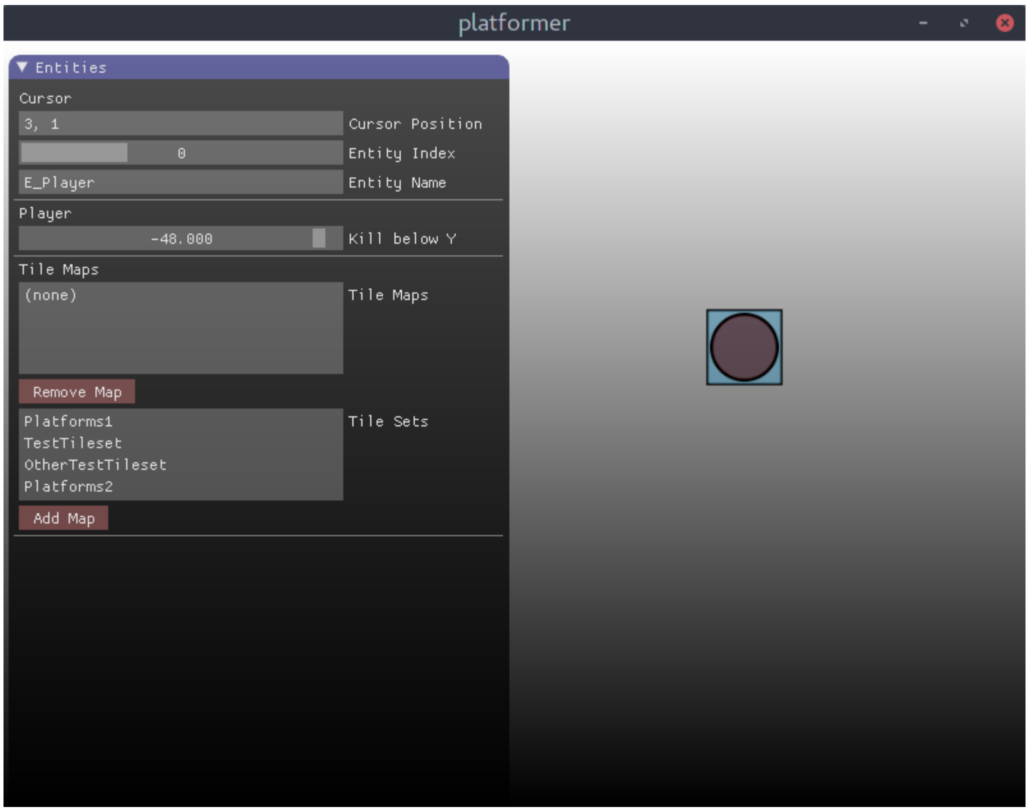
It should look something like this, referring your content where that is applicable. The code below is from the platformer example, which has a very simple player controller that only kills it below a configurable Y value.

```
{
  "E_Player":
  {
    "name": "Player",
    "components":
    {
      "Transform":
      {
      },
      "Sprite":
      {
        "spriteName": "circle.png",
        "renderLevel": 1000,
        "scale": 0.9
      },
      "Physics":
      {
        "bodyType": "static",
        "density": 100,
        "depth": 4,
        "shape":
        {
          "type": "circle",
          "radius": 0.45
        },
        "linearDamping": 10.0,
        "angularDamping": 10.0
      },
      "Noxed::EditorBlueprint":
      {
        "extends": "Player",
        "controlled": true,
        "components":
        {
          "Platformer::PlayerController":
          {
            "args":
            {
              "fallLimit":
              {
                "type": "float",
                "value": -50,
                "min": -1000,
                "max": 0,
                "label": "Kill␣below␣Y"
              }
            }
          }
        }
      }
    }
  }
}
```

Add it to the cursor

At the time of writing, there is no automatic way to locate and index the editor actors. Instead, the object has to be manually added to the Noxed::EntitySpawner component's quickbar on the cursor actor. It should look like the below code example.

One thing to note is that it must match the name at the top of the JSON definition for the editor actor, the file name is not enough. Do not forget to edit that when copying actors.



```
Cursor
3, 1                Cursor Position
0                   Entity Index
E_Player            Entity Name
Player
-48.000             Kill below Y
Tile Maps
(none)              Tile Maps
Platforms1          Tile Sets
TestTileset
OtherTestTileset
Platforms2
```

Figure 3: Editor with your new Player actor

```
"Noxed::EntitySpawner":
{
  "offset": {"x": 0, "y": 0},
  "quickbar":
  {
    "modeName": "quickbarSelect",
    "defaults":
    [
      "E\_Player"
    ]
  },
  "editorMode": 0
},
```

Test it

Assuming nothing went wrong, you now have your player actor selectable, placeable and editable in the editor, similar to how it is depicted on this page.

4.2 Tile Set

In cases like rooms and platforms, an entity is not enough to construct an object of individual shape. Tile sets serve that purpose nicely. A tile set is an image that is segmented into smaller pieces, tiles, that can be arranged in any configuration on a tile map to create a new image.

A tile set needs to point to an image that is loaded along with the atlas, and the following information is needed to properly read the tiles.

- "name": This shows up in the GUI for selection when creating a new map. It should be unique.
- "image": The file name that points to the part of the atlas where the tile set resides.
- "tileSize": The size of an individual tile, in pixels.
- "tileSep": The spacing between each tile, in pixels.
- "setSize": The amount of tiles across and downward.

A tile is then referred to by an index that starts with the top-left corner, and is aligned to the top-left corner.

4.2.1 Structure

The central data structure of a tile map is the `TileGrid`, which essentially is a structure to store a signed integer associated with a two-dimensional position. It has no knowledge of the tile set or tile map it is meant to store data for, and thus doesn't do any range checking on the values. That said, the default value is -1, which is intended to be interpreted as no tile being at that position.

It stores those values in blocks of user-defined size, and associates those blocks with an x index and y index. It also keep them in a linear structure for use in iterating over the entire tile grid to process all tile indices.

This structure uses integers for the positions, which means that reads and writes to it from an actor component need to convert the world position to tile index when operating on the structure.

It is very cheap to iterate over the structure sequentially, and use the block x-index, y-index and size to determine the position of each tile inside of it. The random operations are fast if you access the same block as the last function call, but require two `std::map` lookups with an integer index otherwise.

4.2.2 Smart Tiles

If you have used a basic tile map editor in the past, you may have had to select different tiles of the same component, be it the wall, alternate floor tiles, some flowers to randomize the grass and so on. Additionally, you may have had to cover the tiles up with colliders and other such objects. That is tedious, but predictable work, something that computers excel at.

Another thing is that the tile map layout is sometimes confusing and contain a large amount of tiles for the same wall and floor, and in the beginning there time may be wasted searching for that one top-left convex corner.

That is the inspiration for this feature: a means of setting the tile based on parameters, tags and context. "Smart Tiles" was my working title for this, and it was what stuck. An alternative name may be "Context-Sensitive Tiles", although there is more to this than just context sensitivity.

4.2.3 Guide: Make a tile set definition

This guide covers the process of getting your tile set from the atlas into the editor and your game. There is no programming required to do this part, but you will be limited to the existing tile types – the next guide will cover that.

Step 1: Meta-data and image information

First, you need to take note of some information about your image: it's filename, the size of each individual tile, how much space there is between each tile and how many tiles there are. The tile and separator size is given in pixels, while the set size is given in tiles.

The `"name"` field needs to be unique as to not cause conflict when the tile map requests it. By now, it should look like the JSON below.

```
{
  "name": "UniqueTileSetName",
  "image": "image—name.png",
  "tileSize": {"x": 32, "y": 32},
  "tileSep": {"x": 1, "y": 1},
  "setSize": {"x": 2, "y": 2},
}
```

Step 2: Add smart tiles

For the smart tiles to automate the process of making tile maps, there needs to be information about how it should go about it. The definition for all smart tiles are stored in the JSON file you started on in the previous step.

```
  "smartTiles":
  [
  ]
```

The tile map will refer to them by index, which means any insertions into the middle of the array will potentially break previous smart tile maps.

4.2.4 Tile Types

The `SmartTile` class is abstract, and how it functions is defined by the sub classes which I will describe in the remainder of this section. Each class has a short one-word name that is used to identify it in the JSON tile set definition. The options and arguments are required, unless otherwise specified.

All tile types has tags, which is handled by the base class. They are used for inter smart-tile grouping in some cases, and object spawning. You can also specify conditional tags, which are only returned if the smart tile has been resolved to a specific result at that position. The latter is useful for keeping wall objects from spawning where they are either unwanted or unnecessary.

4.2.5 Guide: Add a new tile type

Here we will go through the process of adding a new tile type in the C++ project, from empty class to a working tile type. In this tutorial, I will implement a simple tile type that uses a secondary tile to form a roof if the above tile is empty, and we'll call it `RoofTile`.

All tile types has to be added to Noxed itself to work, for otherwise it would be limited to the game only and not display correctly in the editor. You could work around that by not using the "dumbDown" option on the tile map and using a dummy tile in the editor, but that will offer a diminished preview of it for the level designer.

The relevant folder is `Noxed/noxed/tileset/tiletypes/` and its corresponding name space is `noxed::tileset::tiletypes`.

The code examples will break style to fit within the horizontal boundaries of the document.

Requirements

- `RoofedTile` uses two tile indices as input, "wall" and "roof".
- It shall have an option called `"emptyOnly"` with default value true.
- If a tile parameter is lacking, they shall get the value -1.
- If the index above matches this instance's index, it shall return the wall tile.
- If the option `"emptyOnly"` is true, return the roof index if there is nothing above it.
- If the option `"emptyOnly"` is false, return the roof index if the tile above has any index different to this `SmartTile` instance's index.
- It has the identifier "roofed".

Step 1: Get a blank slate

The way we use to get a blank slate is to copy the StaticTile header and source file, replace the class name and change the value of the `"ID"` member defined in the source file. Appendix A.1 contains the entire header file for the SmartTile class that our tile will inherit from.

```
// RoofedTile.h
#pragma once

#include <noxed/tileset/SmartTile.h>

namespace noxed { namespace tileset { namespace tiletypes {

class RoofedTile: public noxed::tileset::SmartTile
{
public:
  const static IdType ID;

  void initialize(const std::vector<int>& tiles,
                  const Json::Value& options) override;

  const int resolve(const TileGrid *grid, TileSet *set,
                  const int x, const int y, const int index) override;

  const std::string getType() const override;

private:
  int wallTile;
  int roofTile;
  bool emptyOnly;
};

}}}
```

```
// RoofedTile.cpp
#include "RoofedTile.h"

#include <noxed/tileset/TileSet.h>
#include <noxed/tileset/TileGrid.h>

namespace noxed { namespace tileset { namespace tiletypes {

const SmartTile::IdType RoofedTile::ID = "roofed";

void RoofedTile::initialize(const std::vector<int>& tiles,
              const Json::Value& options)
{
  // The base class sets up tags if they are provided.
  this->SmartTile::initialize(tiles, options);
}

const int RoofedTile::resolve(const TileGrid *grid, const TileSet *set,
                const int x, const int y, const int index)
{
  return -1
}

const std::string RoofedTile::getType() const
{
  return ID;
}

}}}
```

Step 2: `initialize()`

As you see in the code on the previous page, there are two parameters given to the initialize method, `"tiles"` and `"options"`. Those are data from the fields with the same name in the tile set definition.

The first part of the process is getting the tiles from the first parameter, which is an `std::vector<int>` constructed from the JSON array. There is no guarantee that the length will be correct, which means you have to check the size of it before using it.

```
// Set default values
this->wallTile = -1;
this->roofTile = -1;

// Set values from tile array if length is sufficient
if(tiles.size() >= 2)
{
  this->wallTile = tiles[0];
  this->roofTile = tiles[1];
}
else if(tiles.size() == 1)
{
  this->wallTile = tiles[0];
}
```

The next part is easy if you understand the library the NOX Engine uses to parse and export JSON data. As the requirements defined, the default value is false.

```
// Read the option(s)
this->emptyOnly = options.get("emptyOnly", false).asBool();
```

That is all you need to do for initialization.

Step 2: `resolve()`

This function uses the following parameters to resolve the index of the tile that shall be shown at this position.

- grid The TileGrid where the tile is located.
- set The TileSet where this smart tile instance is located.
- x The logical tile x index; xTile parameters in the grid.
- y The logical tile y index; yTile parameters in the grid.
- index The index of this smart tile instance in the tile set.

What we are going to do is to check the index of the tile above it (x, y + 1) in the grid. For this example, I have decided to keep it simple instead of compact. The inner if-statements can of course be replaced with the ternary `?`-operator.

```
int aboveIndex = grid->getTile(x, y + 1);

if(this->emptyOnly)
{
  if(aboveIndex == -1) // If above tile is empty
  {
    return this->roofTile;
  }
  else
  {
    return this->wallTile;
  }
}
else
{
  if(aboveIndex != index) // If above tile is something other than this instance
  {
    return this->roofTile;
  }
  else
  {
    return this->wallTile;
  }
}
```

Step 3: Register it

To register it, open `NoxedWindowView.cpp` and add your header file to the list of includes. It also needs to have been added to `CMakeList.txt` by now, otherwise you will receive linker errors.

The tile set manager is set up in the `NoxedWindowView::initialize` function in the aforementioned CPP file. `TileSetManager::registerSmartTileType` uses the ID field in the template argument to map the short name to your tile type in its instance factory. Add the below line of code among the other calls to the same function.

```
tilesetManager.registerSmartTileType<noxed::tileset::tiletypes::RoofedTile>();
```

Step 4: Test it

Once you have solved any compiler errors and warnings that may have popped up, it is time to test it in order to confirm that it is working as designed.

Find one of the tile set definitions among your assets. For this example, I used tile-set/TestTileset2.json and added the following smart tile definition.

```
{
    "type": "roofed",
    "name": "RoofedTile␣test",
    "tiles": [0, 1],
    "options":
    {
      "emptyOnly": true
    }
}
```

If everything works, adding that tile to a map should appear as depicted below.

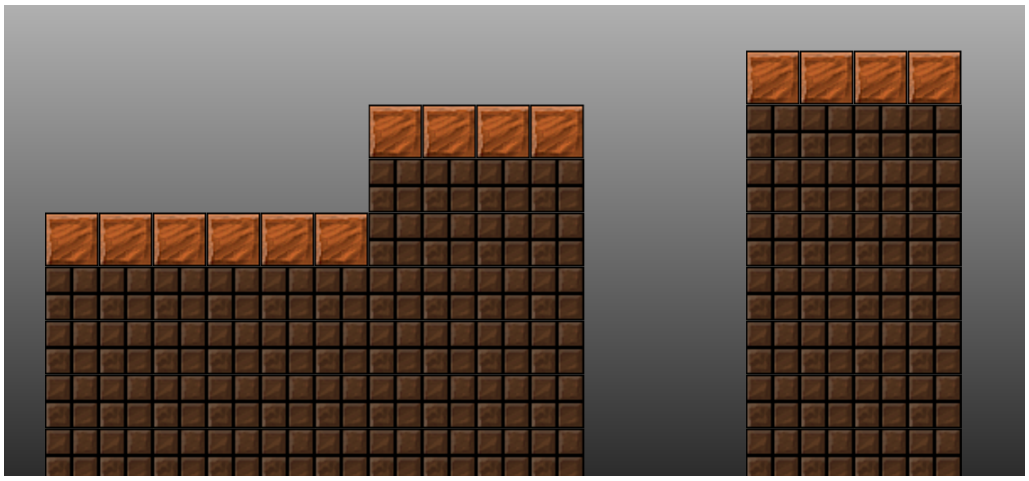


Figure 4: RoofedTile example

4.2.6 Tile Type: StaticTile ("static")

Even with all the features offered by smart tiles, there are still tiles that only consist of one tile from the image, and is never changing based on context. That's what this tile type is for.

Arguments

- 0: The tile.

4.2.7 Tile Type: ChequeredTile ("chequered")

As the name implies, this tile set uses the position to create a pattern similar to a chess board using two tiles from the image.

Arguments

- 0: The white tile.
- 1: The black tile.

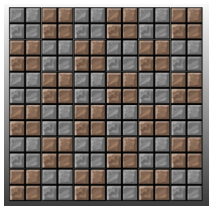


Figure 5: ChequeredTile example (Contrast adjusted for visibility)

4.2.8 Tile Type: BigTile ("bigtile")

This forms a rectangular pattern from multiple tiles, essentially creating one big repeating tile. It is useful for detailed floor textures that are not confined to one square meter, or commonly used patterns. What inspired this tile type was the expensive flooring in The Sims 2 that repeats one bigger texture across multiple floor tiles.

Arguments

- 0-N: The tiles forming the pattern, where N is width time height.

Options

- "width": Width (in tiles) of the pattern.
- "height": Height (in tiles) of the pattern.

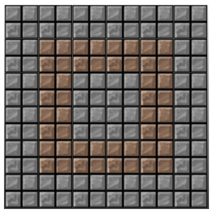


Figure 6: BigTile example (Contrast adjusted for visibility)

4.2.9 Tile Type: RandomTile ("random")

This has a noisy pattern that remain consistent even if the tile is deleted and added again. The random number is a function of the smart tile's index, tile set's name length, x and y.

Arguments

- 0-N: The pool of random tiles, N being an arbitrary number. All elements in this array has equal likelihood, meaning that a more common tile will occur more often.

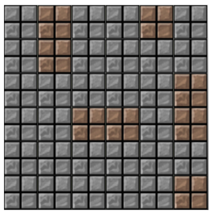


Figure 7: RandomTile example (Contrast adjusted for visibility)

4.2.10 Tile Type: IslandTile ("island")

I struggled to come up with a good name for this one, but what it does is have one outer tile and one inner tile. The inner tile appears if all surrounding tiles are of the same smart tile, while the outer tile occurs if not.

Arguments

- 0: The inner tile.
- 1: The outer tile.

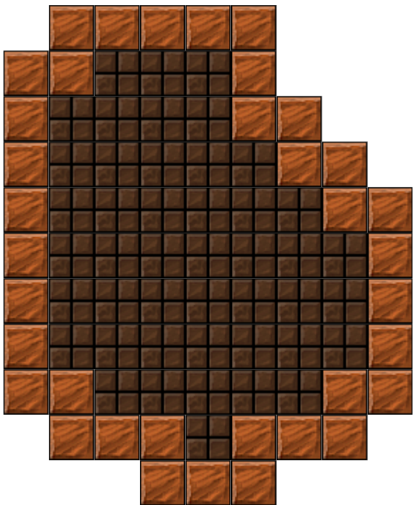


Figure 8: IslandTile example

4.2.11 Tile Type: IslandTagTile ("island_tag")

This is similar to the one described above, but it uses the first tag of its own starting with "tile-group:" as the group, and considers all neighboring tiles with the exact same tag as part of it. It uses the inner tile if all surrounding tags have it, outer tile otherwise.

Arguments

- 0: The inner tile.
- 1: The outer tile.

4.2.12 Tile Type: FourBitTile ("4bit")

This forms a 4 bit mask out of adjacencies and use that to pick the tile index. The adjacencies that affect the bits, in ascending order of significance are: left, top, right, bottom. The only limitations this has is that it doesn't work for T-shapes given that the additional four bits are only used for corners when all four cardinal directions are blocked by a matching tile. The obvious use for this tile is grass platforms, but it could also be used for rocks and paths in a top-down game.

If a tag starting with "tile-group:" exists among its tags, it will check for that instead of the index. Unlike the `IslandTile`, the tile type id is same regardless of whether tags are used.

I recommend keeping the below argument list handy while setting up a tile of this type. If your tile set does not possess all the tiles, I recommend padding it out with the tile you would put at index 15 in the array. If there is no tile at index 15 and the resolved value exceed the array's upper boundary, the first provided tile (no neighbors) will be selected.

Arguments

- 0: No neighbors (0000).
- 1: Neighbor to the left (0001).
- 2: Neighbor above (0010).
- 3: Neighbors to the left and above (0011).
- 4: Neighbor to the right (0100).
- 5: Neighbors to the left and right (0101).
- 6: Neighbor to the right and above (0110).
- 7: Neighbors above and on both sides (0111).
- 8: Neighbor below (1000).
- 9: Neighbors below and to the left (1001).
- 10: Neighbors above and below (1010).
- 11: Neighbors above, below and to the left (1011).
- 12: Neighbors below and the right (1100).
- 13: Neighbors below and on both sides (1101).
- 14: Neighbors above, below and to the right (1110).
- 15: Neighbors all around it (1111).
- 16: Top-left corner unoccupied (1110 1111).
- 17: Top-right corner unoccupied (1101 1111).
- 18: Bottom-right corner unoccupied (1011 1111).
- 19: Bottom-left corner unoccupied (0111 1111).

Options

- *extraBits*: Enables arguments 16 through 19.

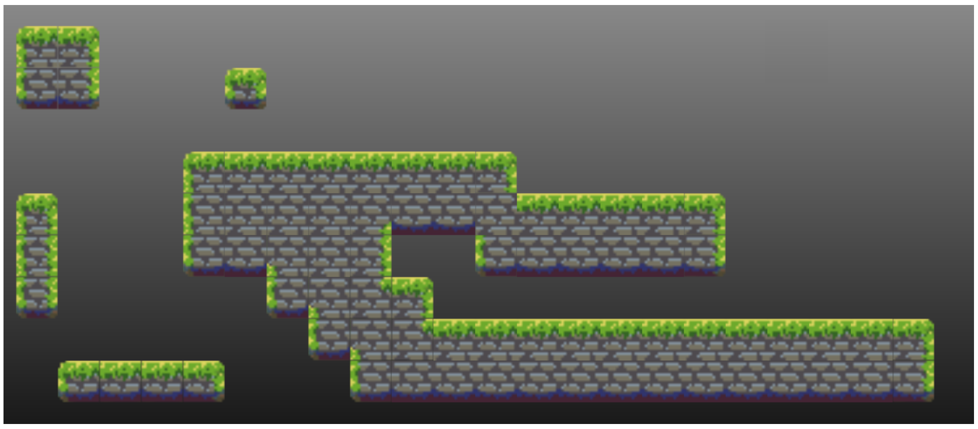


Figure 9: FourBitTile example

4.3 GUI

Our graphical user interface is built with the Dear Imgui library. It is an open source library that outputs vertex buffers for you to render as you please. We chose the library since it was lightweight and very easy to use. The library also have bindings in place to use with OpenGL and SDL. Because of this we were easily able to incorporate the library into our NOX Engine application.

4.3.1 Data Structures and Program Interaction

ElementLayout - A Single GUI Element

The elements of our interface are stored in our program in a struct that looks like this:

```
struct ElementLayout
{
  const std::string key;
  const std::string label;
  const ElementType type;
  Json::Value min;
  Json::Value max;
  Json::Value value;
  std::vector<std::string> listItems;
  bool visible;
}
```

```
enum ElementType
{
  NUMBER_FLOAT,
  NUMBER_INT,
  STRING,
  OUTPUT_STRING,
  BOOLEAN,
  LIST_STRING,
  BUTTON,
  VEC2_FLOAT,
  LAYOUT_HEADER,
  LAYOUT_TEXT,
  LAYOUT_SEPARATOR
};
```

key Identifier used by the the interface to inform an actor which value has been changed.

label Text to be displayed next to the value changed, name of the value.

type Type of value that is being changed and displayed by the interface, text, integrals, float numbers etc. See ElementType.

min Minimum limit of the value if applicable.

max Maximum limit of the value if applicable.

value the actual value.

listItems Used if we are to display an element containing a list.

visible If an UI-element should be displayed or not.

In this figure we see an example of several elements. Looking at the second we can see that it has a **label:** "Brush Size", and a **value:** currently set to 5. Its type is `NUMBER_INT`, which is what we will use to create a slider to change the value of an integer. Its min and max values are 1 and 5 respectively, but are not shown as text. The element with the label "Tiles" is of the `LIST_STRING` type and uses the special *listItems* variable.

4.3.2 The Controller

Which elements are displayed in the user interface is controlled by our *GuiHandler* class. The central data structure used by the class is shown below:

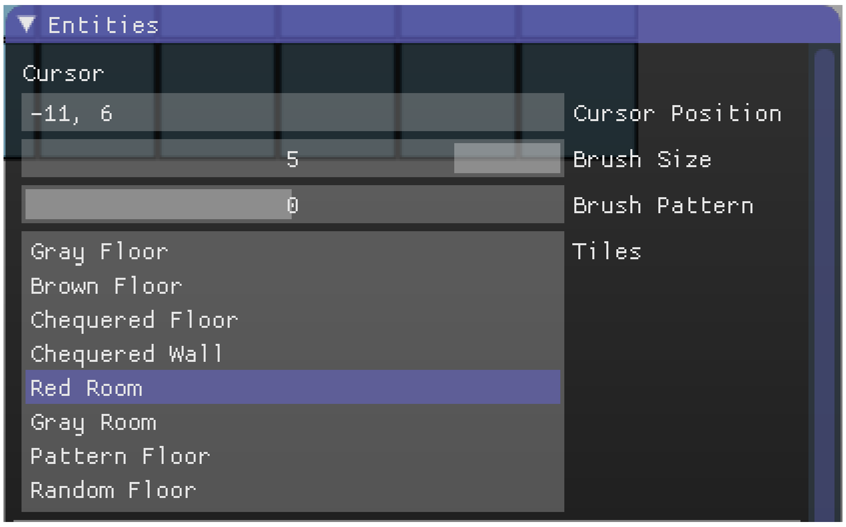


Figure 10: Several ElementLayots

```
struct ElementGroup{
  nox::logic::actor::Actor *actor;
  std::string elementName;
  std::string elementID;
  std::vector <std::shared_ptr<gui::ElementLayout>> layouts;
};

std::map<std::string, std::unique_ptr<ElementGroup>> currentElements;
```

actor A pointer to the actor that owns the values being changed.

elementName Every actor has a name that can be accessed via the pointer. If there is no actor that owns the values the name of the object will be stored here.

elementID An identifier that is shared of actors and objects of the same type.

layouts A vector of `ElementLayout` structs that belong to the actor.

currentElements A map of everything that will be displayed each frame. The value we use for the string key is the `elementID` contained in `ElementGroup`.

The `GuiHandler` will iterate over the `currentElements` map each frame and display the user interface according to what is stored there. The reason we used a map and the `elementID` as a key that we only wanted to show one of each named element group at a time. If you for example place a number of enemies in a row the information displayed by the interface will be the values belonging to the last enemy. You can always click previously placed actors to change their values.

Here we can see an `ElementGroup` being displayed with the name `"E_Enemy"` and several `ElementLayout` elements.

Communication with components

The `GuiHandler` communicates using events. If an actor is placed or selected in the world a `GuiLayoutNotification` event is sent to the interface. In the NOX-engine all events

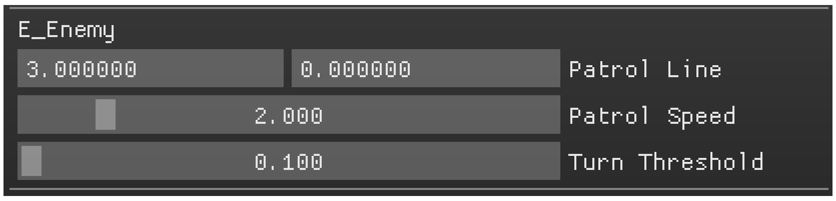


Figure 11: An element group displaying an actor's values.

contain a pointer to an actor, but in addition the `GuiLayoutNotification` event contains the following data:

```
std::string elementId;
std::vector<std::shared_ptr<noxed::gui::ElementLayout>> layouts;
```

elementId The ID that is shared by actors by the same type.

layouts vector of ElementLayout structs that belong to the actor.

The `GuiHandler` will extract the actor pointer and the data from the event and store it in the `currentElements` map. If an element group with the same `elementID` was being displayed, it will be replaced.

Were we to drag a slider or change a value using the interface a `GuiValueChange` event will be sent by the `GuiHandler` to the actor that was associated with the element. The `GuiValueChange` event like all other events contains an actor pointer and this information:

```
std::string elementId;
std::string key;
Json::Value value;
```

elementId The ID that is shared by actors by the same type.

key Identifier for which value is to be changed.

value The actual value.

4.3.3 Guide: Setting up GUI for a component

There are a few steps you need to make to get a component within the editor to communicate back and forth with the GUI handler. For this, I will describe the process of making it possible to set the size of the grids `GridCursor` (4.4.5) snaps to. This guide assumes you are familiar with the event system, header files and name spaces, thus it will not go into detail about it.

Step 1: The data structure

The best practice for storing UI elements are to keep persistent instances of them in the component, and sending the shared pointers when the UI element group needs to be updated. That way, the component can manage the elements' state. How we have done it is having an array that is populated once and sent off every time we need to update the GUI.

```
std::vector<std::shared_ptr<noxed::gui::ElementLayout>> layouts;
std::shared_ptr<noxed::gui::ElementLayout> gridSizeLayout;
```

Step 2: Initialize

Start by making a function for initializing the GUI element layouts, and have it called by the creation event function. It should construct the element, add it to our list, and then call for an update. While it does not make a lot of sense for a small GUI element group like this, it makes it easier to read code setting up larger element groups, especially where elements require loops and conditional statements.

The GUI element should have a key that can be understood in the code, a label that makes it clear to the user what it does, and a type that fits with its purpose. The other fields are not relevant right now, as a `VEC2_FLOAT` field's limits has to be enforced and defined when receiving value changes.

```
void GridCursor::initGuiLayout()
{
  // Create element
  this->gridSizeLayout = std::make_shared<ElementLayout>("gridSize",
                                                          "Grid␣Size",
                                                          ElementType::VEC2_FLOAT,
                                                          Json::Value());

  // Add it to list
  this->layouts.push_back(this->gridSizeLayout);

  // Update it
  this->updateGuiLayout();
}
```

Step 3: Update

Every time a value changes, even if it's caused by the GUI itself, it needs to get up-to-date information.

```
void GridCursor::updateGuiLayout()
{
  // Update grid size
  this->gridSizeLayout->value["x"] = this->gridSize.x;
  this->gridSizeLayout->value["y"] = this->gridSize.y;

  // Update position
  const float x = this->actorTransform->getPosition().x;
  const float y = this->actorTransform->getPosition().y;
  this->positionLayout->value = boost::lexical_cast<std::string>(x) + ",␣"
                              + boost::lexical_cast<std::string>(y);

  // Notify GUI Handler
  auto event = std::make_shared<GuiLayoutNotification>(this->getOwner(),
                                                        "GridCursorOptions",
                                                        this->layouts);

  this->getLogicContext()->getEventBroadcaster()->queueEvent(event);
}
```

Step 4: Listen

The GUI will now display the element as two adjacent text inputs. However, it will not do anything with the grid cursor in response to user input. To handle that, the component needs to listen to the `GuiValueChange` and handle it as shown in the code snippet below.

```
else if (event->isType(GuiValueChange::ID))
{
  auto valueChange = std::static_pointer_cast<GuiValueChange>(event);

  if(valueChange->getKey() == "gridSize")
  {
    this->gridSize = parseJsonVec(valueChange->getValue(), glm::vec2(0.f, 0.f));

    auto position = this->actorTransform->getPosition();
    this->actorTransform->setPosition(this->snap(position));

    this->updateGuiLayout();
  }
}
```

Step 5: Test

Once it compiles, you should see a GUI element with the name of the actor as header. It is supposed to look like the figure below. As you edit it, it should automatically snap the cursor's position to the new grid.

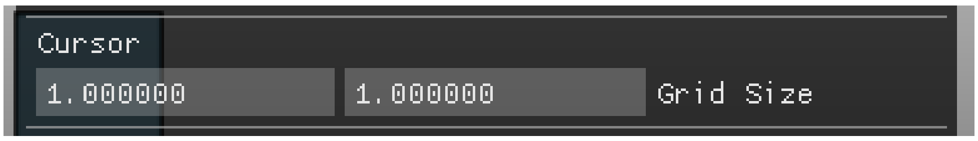


Figure 12: Grid size UI element

4.4 Noxed Components

All Noxed components have a `Noxed::` prefix in their identifiers used to include them in actors, and can thus be easily identified in the JSON files for their definitions. Similarly, they use the C++ name space `noxed::components` for their declaration.

4.4.1 Brush

The brush manages the pattern and size of tile placement in the world grid. The patterns are classes that defines the set of points that form the core and edge of the brush, as they do not have the same requirements when it comes to resolving smart tiles.

While it reacts to mode changes from keyboard and UI input, it does not interact directly with the world. Neither does it manage the UI elements. It exposes functions that `TileSpawner` (4.4.10) uses to create the user interface element, and to read the array of offsets to figure out where it should alter the tile map.

It makes use of the singleton `BrushPatternManager` to get the registered brush patterns. The singleton is set up during the windows' initialization as shown in the code snippet below.

It also has its own renderer that renders the brush sprite for every point that the brush pattern has for that size. It allocates enough squares to use the max size of the brush.

```
brushPatternManager->registerPatternType<noxed::brush::patterns::SquareBrushPattern>();
brushPatternManager->registerPatternType<noxed::brush::patterns::DiamondBrushPattern>();
```

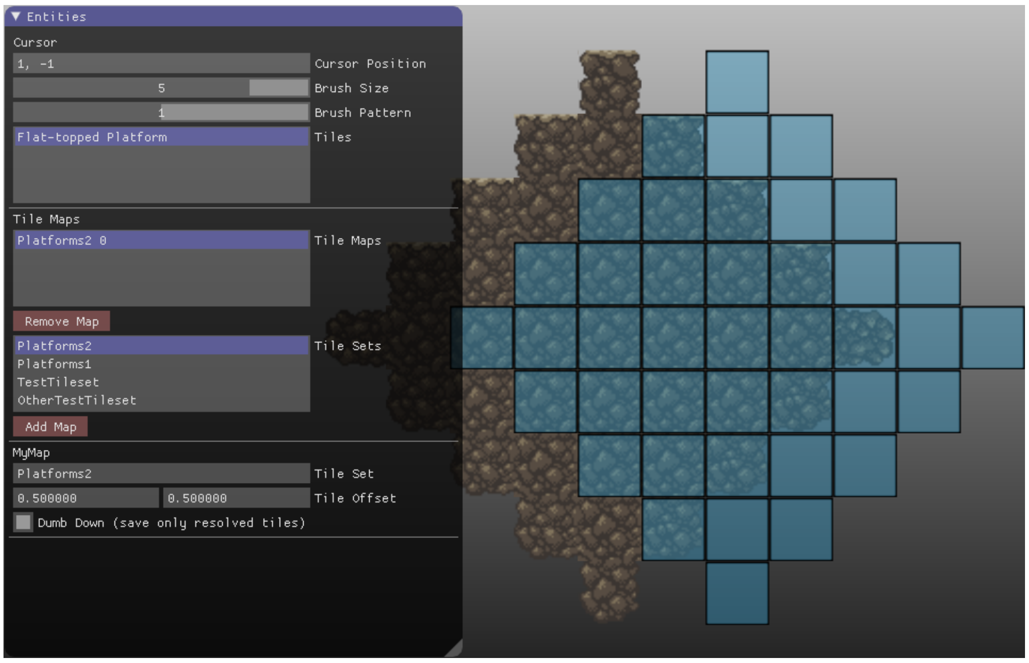

```
▼ Entities
Cursor
1, -1                      Cursor Position
5                          Brush Size
1                          Brush Pattern
Flat-topped Platform       Tiles
Tile Maps
Platforms2 0               Tile Maps
Remove Map
Platforms2                 Tile Sets
Platforms1
TestTileset
OtherTestTileset
Add Map
MyMap
Platforms2                 Tile Set
0.500000   0.500000        Tile Offset
Dumb Down (save only resolved tiles)
```

Figure 13: Brush example picture.

4.4.2 EditorBlueprint

This component is the primary means of saving an actor when the level designer initiates a world save. Its only dependency is Transform, but it could have others if configured to use the editor component when serializing it. This may seem contradictory to section 4.1 where it is mentioned that physics and any graphical component is necessary; however, that is only when the object has to be clickable and visible. If that is not required, it can be used with only a transform to make an object that is included in exported levels, which the example below shows.

Example

```
{
  "components":
  {
    "Transform":
    {
    },
    "Noxed::EditorBlueprint":
    {
      "extends": "LevelController"
      "components":
      {
      }
    }
  }
}
```

4.4.3 EntitySpawner

This component is responsible for spawning, selecting and deleting entities. It does not move around on its own, and is thus expected to be in an actor that has a `GridCursor` (4.4.5). It emits an `EntitySelection` (4.5.1) event that is picked up by the correct `EditorBlueprint` (4.4.2) to display their options in the user interface.

As mentioned earlier in this chapter, this object keeps a list of which entities it can spawn.

4.4.4 FollowCamera

This component, when on a controlled actor, changes the properties of the `NoxedWindowView`'s camera. It tracks the position and rotation of the parent actor's `Transform`, and changes the camera according to configuration. It is used on the player character in the example games.

4.4.5 GridCursor

As its name suggests, this is a component that makes the object move around according to the cursor's position while snapping to a grid that is set in the component definition. It listens for `MouseAction` (4.5.4) which are sent by the window on both mouse and camera movement.

It does not possess any functionality for acting upon the editor world in and of itself, instead relying on the `EntitySpawner` (4.4.3) and `TileSpawner` (4.4.10) component for that.

4.4.6 PushCamera

This is similar to `FollowCamera` (4.4.4) in that it controls the `NoxedWindowView`'s camera, but it uses movement controls and pushable edges to move it. It is designed for the `GridCursor` (4.4.5). The second means of control, which is by pushing the cursor against the edges of the screen, is implemented but left unused as it became too much of a nuisance in practice.

4.4.7 TileMap

This component is a canvas, upon which tiles from a tile set can be arranged in any configuration. It can both be a simple tile map that just uses the indices provided to it, or a smart tile map that resolves the smart tiles before displaying it. It manages the tile grid and the render nodes necessary to display it.

If the tile map is set to use smart tiles, any set operations on it will go through the smart tiles at the specified indices and resolve them. It will only resolve the placed tiles and its eight adjacent tiles. For brushes larger than one cell, the brush instructs which part of it is the core where only the tile itself has to be resolved, and which positions are the edge where tiles are to be resolved along with their neighbors.

If you haven't already, I recommend reading section 4.2 which goes into detail about tile sets and smart tiles.

When initialized without being provided a tile set, it will send out a `TileSetRequest` (4.5.6) and wait until receiving a response before setting up the renderer.

TileBlockRenderNode

To display the tile maps, we had to create a custom render node that would show one square for each tile in the map. The tile map component has one renderer per block in the `TileGrid`, which can be of any size. This means that a change in the tile map will cause as few nodes as possible to update, sometimes even only one.

One problem with the renderer is that it will exhaust the engine's limitations after approximately 10,000 tiles has been placed, and even less than that if the tile map is sparse.

4.4.8 TileMapManager

This component manages the lifetime of multiple tile maps that exist in the game world, and offers a user interface for the level designer to do so. The tile maps themselves do not have any physical presence in the editor world, making this the only way to remove or select them. It also shows which tile sets are available.

4.4.9 TileMapObjectSaver

A useful feature for smart tile sets are spawning actors at the tiles' location. The obvious use case is spawning solid collision blocks onto tiles depicting solid objects in order to avoid the second step of covering them with black collider boxes. They use tiles' tags to determine whether an object is to be placed at the given position. Please note that it does not check whether the grid cell is occupied before doing so.

With the aid of conditional tags, multiple objects can be used for different tile shapes. An example would be saving memory and storage space by having the colliders only on the outward facing tiles. It could be more granular than that, and even allows for different actors depending on what the smart tile resolve into - triangular ones on corners, for example.

This component needs to be in the same actor as the `TileMap` (4.4.7) it's supposed to overlay with actors.

Example

```
"Noxed::TileMapObjectSaver":
{
  "objects":
  {
    "wall:square": "InvisibleWall.Square"
  }
}
```

4.4.10 TileSpawner

This component allows for managing the content of a single `TileMap` (4.4.7) by placing or removing tiles according to the size and pattern of the actor's `Brush` (4.4.1). It does not interact directly with the tile map, instead it sends a `TilePlacement` (4.5.5) that the tile map will react to.

4.4.11 WorldSaver

This component waits for its configured key combination, and then sends out a `WorldSaveBroadcast` (4.5.7)] that other actors' components can listen to in order to write to the world file, and the `NoxedWindowView` attached the same work in progress object to a `WorldSaveResponse` (4.5.7) that is received by this component after the request has reached all listeners. The only Noxed components that do so are `EditorBlueprint` (4.4.2), `TileMapObjectSaver` (4.4.9) and `TileMap` (4.4.7).

The default configuration is to save this to a file called "initial.json", and there is at the time of writing now means of changing that during runtime, and the default key combination is left CTRL and S.

4.5 Noxed Events

4.5.1 EntitySelection

This is a very simple event that has nothing more to it than the ID and base class. It signifies that an entity has been selected by the `EntitySpawner` (4.4.3). Any component listening to this should mark themselves as selected if, and only if, the actor pointed to is their parent, and mark themselves otherwise if not. This is to prevent many `EditorBlueprint` (4.4.2) components fighting over the UI element for the blueprint's properties.

4.5.2 GuiLayoutNotification

The UI, which is described in detail in section 4.3, needs to be populated by the actors before it can be used. Given that UI is segmented into element groups that are associated with an actor, the relevant components can use this event to notify the UI controller about who they are, what element group they want to manage and what elements there are in that group.

See section 4.3.2 for more information about this and the below event.

4.5.3 GuiValueChange

As the communication from actors to the user interface is done via events, so are the replies. This contains the name of the actor that described the UI element group, the name of the group, the key that was supplied with the changed element, and last – but not least – the value. The value is a `Json::Value` meaning that it is up to the components to convert them to the correct type, which should happen without error if the UI controller received the correct element types.

4.5.4 MouseAction

This event is for any and all actions based on mouse input. It differs from all other NOX Engine and Noxed events in that it has 3 identifiers that are all mutually exclusive, `MouseAction::MOVE`, `MouseAction::BUTTON` and `MouseAction::HOLD`. It is possible to listens to more than one of them, even all three. The reason for the separation is to prevent unnecessary processing of events that are potentially ran on every frame.

The following data is broadcast along with the event:

- Position on screen (in pixels).
- Size of the screen (in pixels).
- Absolute world position of the curosr (in game units).
- Whether it's a button press in case of a `MouseAction::BUTTON` id.
- Which button is pressed.

The event also triggers a new mouse movement event when the camera is moved, with a recalculated world position. This prevents the `GridCursor` (4.4.5) actor from lagging behind if only the camera moves without the mouse moving. That allows the level designer to place tiles in straight lines by only moving with the keyboard while holding down the mouse button.

4.5.5 TilePlacement

The `TileSpawner` (4.4.10) only knows the name of the tile map, and don't have any controls or direct reference to the actor that owns it, or the component itself. To communicate with the `TileMap` (4.4.7), it sends these events that addresses a tile map by name. The event contains the position, brush pattern, brush size and the desired tile index.

Another use of this would be in a game where a smart tile set is to change due to a game mechanic. For example, we can have an actor that overlay all cracked destructible walls, and upon destruction it will tell the tile map to change the tile at its position to reflect the changed state.

4.5.6 TileSetRequest and TileSetResponse

Tile maps not managed by the `TileMapManager` (4.4.8) have no direct access to the manager that keeps track of all tile sets. A `TileMap` (4.4.7) starts by sending this event to the window, which accesses the tile set manager and constructs a response to provide the appropriate tile set data to the map.

A secondary purpose of it is to get a reference to the aforementioned tile set manager as it is included in all responses. That is what `TileMapManager` (4.4.8) does in order to supply tile maps with the right set.

Request and response events is something often used in the NOX Engine to get data from other actors, as well as references. That way, the numeric identifiers need not be hard-coded into the levels, nor would a pointer to the window or application be necessary. The camera controllers described earlier in this chapter was made prior to us learning about this pattern of inter-actor communication.

4.5.7 WorldSaveBroadcast and WorldSaveResponse

Our first idea of saving the world involved having all created objects be a child of the cursor actor, which was unnecessarily complicated and limited all editor actors to one parent actor. After better understanding the event handling, we switched to a process where every object that wanted to could extend the world root or add another actor to the array of actor definitions.

This meant that multiple types of components could listen to the broadcast and add actor definitions or world properties as they see fit. Below are three components in our project that uses this event, and you can read more about them in their sections.

- `EditorBlueprint` (4.4.2)
- `TileMap` (4.4.7)
- `TileMapObjectSaver` (4.4.9)

This is unlike the above request/respone pattern as the response's purpose is only to be next int the event queue. Upon receiving this response, the `WorldSaver` (4.4.11)will be aware that all objects listening to the previous event has had a chance to save data to it, and it is thus ready to write the resulting world file.

5 Deployment

This chapter discusses how to prepare and package Noxed for use by end users, as well as how to set up a development environment for projects using or working on Noxed.

5.1 Development environment setup

5.1.1 For Windows

Before you begin you need to install these programs on your computer:

- Visual Studio 2015 with c++ languague pack.
- Git
- Cmake (with GUI)

Large parts of this section is copied almost verbatim from the NOX Engine readme at `https://bitbucket.org/suttungdigital/nox-engine`, with some clarifications. This guide will use the precompiled versions of the third party libraries available from the Suttung Digital Bitbucket repository.

Setting up Third Party Libraries

Navigate to where you want to put the project and open Git Bash there. We will need to create a folder for our projects and clone the libraries. Run these commands in Git Bash:

```
mkdir Noxed
cd Noxed
git clone https://bitbucket.org/suttungdigital/windows-libraries
```

In windows explorer navigate to the windows-libraries folder and run the script named "`create_msvc140_usr`" (right-click -> run with powershell). You might need to allow running scripts on your system. To do this, open a PowerShell terminal and run

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

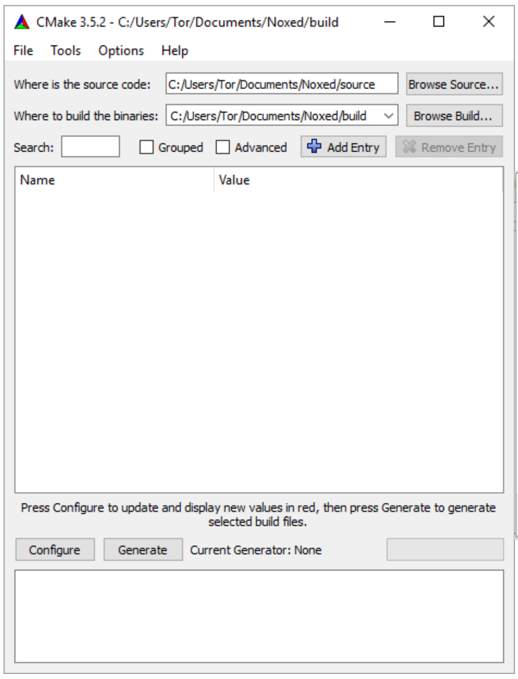
When the script is done, there will be a new "`usr`" directory next to the script. Set the the environment variable "`CMAKE_PREFIX_PATH`" to point to this. Also set "`PATH`" to point to "`usr\bin\x64`" and "`usr\bin\x86`" to be able to use both the 64-bit and 32-bit dll's.

Setting up Noxed

Browse to your Noxed folder in Git Bash. We will now clone the Noxed repository into a folder named source and create a build folder for our Visual Studio solution.

```
mkdir build
git clone https://bitbucket.org/noxedunited/noxed source
cd source
git submodule update --init --recursive
```

Start the Cmake program. Press the "`Browse Source`" button and select the "`source`" folder where you cloned the Noxed repository. After that press the "`Browse Build`" button and navigate to the "build" folder.



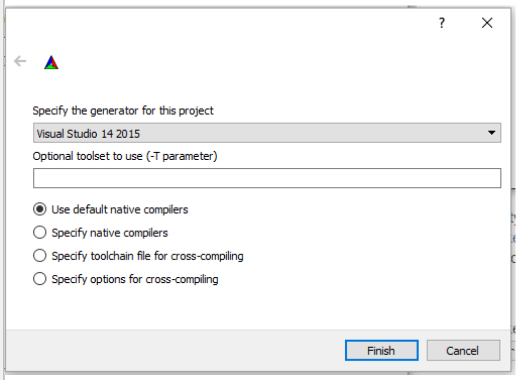
Where is the source code: C:/Users/Tor/Documents/Noxed/source

Where to build the binaries: C:/Users/Tor/Documents/Noxed/build

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Figure 14: Source and Build folders for Cmake



Specify the generator for this project: Visual Studio 14 2015

Optional toolset to use (-T parameter)

- [✓ Use default native compilers]
- Specify native compilers
- Specify toolchain file for cross-compiling
- Specify options for cross-compiling

Figure 15: Specify Visual Studio for Cmake.

Press Configure and a dialogue box will open. Select "Visual Studio 14 2015" from the dropdown menu and tick the box for "Use default native compilers", press Finish. After this press Configure followed by Generate. If you get an error that glew is not found when Configuring try to set the "`CMAKE_PREFIX_PATH`" to "`...\usr\lib`" instead of "`...\usr`".
You should now have a ready to run solution inside your "`\Noxed\build`" folder. The three different projects you can run are:

- noxed-editor
- topdown
- platformer

Right Click your preferred project, select "`Set as StartUp Project`", then click on Properties->Configuration Properties->Debugging and set the Working Directory to your source folder, typically "`..\..\source`". If you want to run the noxed-editor on the protogames you need to supply either "topdown" or "platformer" as command arguments.

5.1.2 For Linux

All the commands below are run as a non-privileged user unless `sudo` is used. All commands also assume you are using Ubuntu 14.04 LTS with brief explanations on how it differs on later versions ov ubuntu. If you run a distribution other than Ubuntu, the package manager commands will differ greatly. Furthermore, if you use another shell than `bash` or plain `sh`, you may need to make some changes.

First you need to get the dependencies and tools to compile and run the package. The below command is for Ubuntu 14.04 LTS and may differ on your Linux system. The first line is only requires on Ubuntu 14.04 LTS to get a GCC version that supports C++14 features used in NOX Engine. For later versions of ubuntu, you may want to replace g++-5 with simply g++

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install git cmake g++-5 libboost-all-dev libglew-dev
```

Once that is done, you should make a folder for Noxed and clone the repository. After that, you will need to grab the NOX Engine and its submodules.

```
mkdir noxed
cd noxed/
git clone https://bitbucket.org/noxedunited/noxed source
cd source
git submodule update --init --recursive
```

If everything went according to plan, building it should be as simple as running CMake and make in the build directory. Omit the option for cmake if your compiler's command is simply g++.

```
mkdir ../build/
cd ../build/
cmake ../source -DCMAKE_CXX_COMPILER=g++-5 && make -j2
```

To test it, navigate to the source folder and run one of the executables.

```
../build/bin/noxed-editor platformer
../build/bin/platformer
../build/bin/noxed-editor topdown
../build/bin/topdown
```

5.2 Packaging

5.2.1 For Windows

For this guide to work you need to have the same folder structure as the development environment setup guide.

- In Visual Studio, select Release instead of debug and compile the three projects.
- Go into your source folder and run the script named `win_release.ps1`(Right-Click -> Run with Powershell), you may need to set permissions to run scripts first. The script will create a release folder in your Noxed directory
- Download a Microsoft Visual C++ 2015 Redistributable package that corresponds to your version of the Visual C++ compiler and put it in the release folder. At the time of writing this is 14.0.23026 and can be found at https://www.microsoft.com/en-us/download/details.aspx?id=48145[10]

Your release folder is now standalone and can be run on other windows computers. Remember to install the Microsoft Visual C++ 2015 Redistributable before you run the programs.

5.2.2 For Linux

To deploy our two example games and the editor, I have prepared a shell script that will run under Ubuntu 14.04 LTS. It should be put in the `noxed/` directory mentioned in the last chapter, which is above the source and build directory. You can find the script in the root directory of our repositry. The script has to be run from the directory outside of the source directory that Noxed was cloned into.

The result is two folders, one for each game. They contain the editor and a shell script to run the editor with the correct command argument to load the resources. This script does not package them for installation, however and they will have to be run from the folder.

5.3 Licensing issue

At the time of writing, the executable produces will be subject to the LGPLv2 license due to statically linking OpenAL. The NOX Engine developers are looking into making them shared by default and thus our script and instructions may need modifications to work properly in the future. To compile OpenAL dynamically or omit it, you need to change the options in NOX-Engine's `CMakeLists.txt` file that are relevant to it.

6 Discussion

6.1 Cross-Platform Development

Developing for both Linux and Windows simultaneously went surprisingly well. After configuring Visual Studio to use forward slashes we had zero instances of code written in Visual Studio not running on Linux and only two instances of Linux written code not running on Windows. We believe that this is mostly due to the NOX Engine already being cross-platform.

Our only third-party outside of the NOX Engine library was written with both Linux and Windows in mind, but required a tiny bit of extra code to run under Windows.

Listing 6.1-1: The only platform specific code.

```
#ifdef _WIN32

  SDL_SysWMinfo wmInfo;
  SDL_VERSION(&wmInfo.version);
  SDL_GetWindowWMInfo(this->window, &wmInfo);
  io.ImeWindowHandle = wmInfo.info.win.window;
#endif
```

One major issue we had was that the Graphical User Interface would not show up on screen when the editor was running on a Linux machine with an integrated graphics card. After spending some time investigating this we decided to cut our losses and not support integrated cards on Linux.

6.2 Code Style and Quality

https://github.com/isocpp/CppCoreGuidelines[11]
Our original idea was to use the c++ core guidelines. This was quite quickly discarded. We were impatient to start working and constantly looking up something was very frustrating. Instead we decided to imitate the style and practices used in the NOX Engine and its examples. This proved to be a more comfortable process since we often had to check the engine code anyway.

6.3 Workflow and Process

We got all our planning done in time and started programming the 26th of January.

Although our plans were to meet daily in the same room we ended up using our office more as a place for organizing work than a workspace. Preferring to work at home we met at least a couple of times face to face per week while keeping contact on Skype and Slack to coordinate our work. If there was any issues or we needed feedback or close cooperation we would use Skype voice chat and work together that way.

Using sprints as a container for issues worked reasonably well. We think that if we were to do a similar project we would try using a Kanban agile model to see if the increased flexibility would give us bigger freedom to refactor what work needs to be done next.

The goals were coming along as scheduled for the first two milestones, but the tile set and UI content proved to be more work than first thought. We were threading on unfamiliar ground with NOX Engine's rendering mechanisms. Suttung Digital were very helpful with explaining it, however, otherwise it could have taken a lot longer. In hindsight it might be easier designing and implementing a graphic user interface simultaneously as the other mechanics.

Come April and the sickness risk started to became reality. That ended up – along with what I mentioned above – causing a 2.5 week delay. It gave us less time to polish and improve usability of the editor, as well as doing code related to the game prototypes.

6.3.1 Jira and Git

In hindsight we should have found a simpler tool with simple issue tracking where you could add issues to sprints. That was all we really needed and wanted. It didn't help that the web interface was updated in the middle of our project. We managed to use Jira for all of our issues and work so that we had total control of who was working with what when.

Each issue got a separate branch as planned with the naming convention "wX-noxed-Y" where X was the week number and Y was the issue number. Later we switched to "NOXED-Y" since it was simpler and the week the issue was created didn't really matter. Code reviews before every merge into master weren't accomplished, but we managed to give each other feedback on coding style frequently.

Git and Bitbucket worked flawlessly with one small exception; submodules. After moving the location of one submodule we never managed to remove all traces of the old location, making it pop up as a change periodically.

6.3.2 Suttung Digital

Having Suttung Digital as our principal was a good experience. We had weekly meetings with them, except for one week in May and the easter holiday. The meetings were a very good incentive to keep our work consistent and on time, in addition to getting some very valuable feedback from Suttung Digital.

They added us to their Slack server which gave us a way to communicate with them directly, providing us with prompt and constructive answers on any and all questions we had. They even answered us on weekends, which was a pleasant surprise.

6.3.3 Future Work

There are a few things that will need to be done before the editor can become a great tool for development within the NOX Engine. The project will be made open source with an MIT license, meaning that any of these issues may no longer be there when next year's Bachelor groups are coming together. We will continue managing the repository hosted on BitBucket as of the time of writing, and submissions from outside our groups will have to be submitted as pull requests.

To see the project at the state it was upon delivery, we have created the tag "bachelor-2016-hand-in" at the last commit we did before delivering the source code.

Refactor to library

As it stands, the project includes the source code for Noxed and the games in the same repository. This is not very helpful for any new projects that wants to use it as their project will have to be forks or submodules of Noxed. The tasks are as follows:

- Change compilation type to static library
- Move `platformer/` and `topdown/` to their own repository with Noxed as submodule
- Write instructions for how to set up a project
- Add a means of extending the editor in projects using Noxed.

All editor-only components should still be mixed in with the rest of the Noxed components, and that is to allow it being used in future projects where the level editor is embedded into the game. It is entirely possible to make a constructor argument or function to select whether they should be part of the current running Noxed program, but we never saw the need for it, or a case where extra components would impact performance.

Save and resume progress

This is a more complicated issue, but is crucial to making this viable in a production environment. There is currently no way to save the world in a way that can be restored in the editor for future work. What it needs to do is to serialize all active objects aside from a select few that may change during development, like the cursor and any other persistent `EditorBlueprint` (4.4.2) actors.

File managment

The files written to and read from in the editor are all statically configured in the relevant components' JSON files. This editor needs a way to invoke the operative system's file dialogs and select files from there for load, save or export. Either that, or a user interface that allows adding, removing and selecting files within the editor.

Better TileMapObjectSaver

The object saver is useful for placing walls and any other object that functions independently of the tile map itself. However, what if we for example wanted to make every cracked wall destructible and parameterize it so that it knows which tiles to put down instead when the wall is destroyed.

A way to go about it is having a specific component that is saved with any actor definition this component produces, and within it there has to be information about what tile indices to change into. It also needs to differ depending on whether the tile map is "dumbed down" (i.e. only resolved indices are saved, not the smart tile indices) as the desired index would be different in that case. This `SmartTile` base class would need to store the relevant information.

Blueprint argument visualization

A feature to visually aid the level designer would be line-art squares and circles affected by arguments in editor blueprint, for example a circle to depict light range.

A proper game

Our two games are mere prototypes of basic mechanics in their genre, and do not represent a real use scenario for the editor. This is probably the only thing here that may unconditionally serve as a starting point for a future bachelor project: developing a game while contributing relevant changes upstream to the editor.

7 Conclusion

As we have discussed in the previous chapter, the project reached its goal and we can successfully create levels for projects using the NOX Engine with the Noxed extensions as our two prototypes will show. Even though the editor is done as per our original goals, and we are happy with how it turned out, we discovered a lot of room for additional features and improvements in order to make it the go-to level editor for NOX Engine projects.

In the process, we learned a lot about working with other open source software. That means perusing code to find out how something works when documentation does not suffice, and most importantly contributing back when there is something that needs to be added or fixed to fit our use of it.

Our group also gained experience in planning and using an agile development model, discovering the strengths and weaknesses of it in the long term. It worked well with our group where the tasks were set from consensus and input from Suttung Digital, but we can not tell how well it scales were the project larger than two people. Our milestones served well as an indication of the project's status.

Bibliography

[1] 2016. Suttung digital as. http://suttungdigital.com/. (Visited March. 2016).

[2] 2016. Git. https://git-scm.com/. (Visited January. 2016).

[3] 2016. Jira software. https://www.atlassian.com/software/jira. (Visited January. 2016).

[4] 2016. Slack. https://slack.com/. (Visited January. 2016).

[5] 2016. Toggl. https://www.toggl.com/. (Visited January. 2016).

[6] 2016. Cmake. https://cmake.org/. (Visited January. 2016).

[7] 2015. Visual studio. https://www.visualstudio.com/. (Visited November. 2015).

[8] 2016. Atom. https://atom.io/. (Visited Januar. 2016).

[9] 2016. Texstudio. http://www.texstudio.org/. (Visited April. 2016).

[10] 2016. Microsoft visual c++ 2015 redistributable. https://www.microsoft.com/en-us/download/details.aspx?id=48145. (Visited April. 2016).

[11] 2015. C++ core guidelines. https://github.com/isocpp/CppCoreGuidelines. (Visited December. 2015).

A Header Files

A.1 SmartTile.h

The file has been edited slightly to break very long function declrataions so that they do not exceed the box width.

```
#pragma once

#include <json/value.h>

#include <noxed/tileset/TileGrid.h>
#include <string>

namespace noxed { namespace tileset {

class TileSet;
class TileGrid;

/**
* A smart tile type. This class is to convert a smart tile to a regular dumb
* tile by using the arguments it is provided.
*
* There is one instance per smart tile in a tile set. It should not keep state
* that is relevant to a single tile map.
*/
class SmartTile
{
  public:
  using IdType = std::string;

  SmartTile() = default;

  /**
  * Initialize the smart tile with
  *
  * @param tiles A list of tile parameters
  * @param options A json object with options
  */
  virtual void initialize(const std::vector<int>& tiles,
                          const Json::Value& options);

  /**
  * Resolve the current tile into the underlying tileset tile.
  *
  * @param grid The TileGrid where the tile is located
  * @param set The TileSet where the smart tile instance is located
  * @param x The logical tile x coordinate; xTile parameters in the grid
  * @param y The logical tile y coordinate; yTile parameters in the grid
  * @param index The index of this smart tile in the tile set.
  */
  virtual const int resolve(const TileGrid *grid, const TileSet *set,
                            const int x, const int y, const int index) = 0;

  /**
  * Ask if the tile has a tag. This is for checking instances across instances
  * of smart tiles. For example, you may want two different wall tiles to be
  * context sensitive.
  *
  * If your tile class do not need this, return false without any other checks.
  *
  * @param tag The text string to ask it for.
  * @param resolvedIndex The index resolve() has given, or -1 to not bother
  */
  virtual const bool hasTag(const std::string tag,
                            const int resolvedIndex) const;

  /**
  * Check if the tag is at all possible for this smart tile.
  *
  * @param tag The tag word
  * @return True if the tag is possible, were the right conditions met
  */
  virtual const bool canHaveTag(const std::string tag) const;

  /**
  * Get the type string that should be the constnat ID.
  */
  virtual const std::string getType() const = 0;

  protected:
  std::map<std::string, std::vector<int>> tags;
};

}}
```

B Original Project Plan

1. Mål og Rammer

1.1. Background

We are two students studying Game Programming at NTNU Gjøvik. Our wish is to create a level editor for a game made with the NOX-Engine made by Suttung Digital. Our main focus is to get an editor that can be used to generate levels without any need to change the game.

We want to have support for tilesets, entities and solid textured objects.

1.2. Project Goals

Effect Goals:

- Get a deeper understanding of how games and level editors interact.
- Learn more about tool development. Creating tools that are practical and usable.

Result Goals:

- The editor should be usable and complete.
- The basics of the editor should be understandable and intuitive.
- The levels created should work on the game without changing or modifying the game.
- The games and the editor should work in both Windows and Linux.

1.3. Project Framework

The game(s) and the editor will be made using the NOX engine. We will use free code editors. If we need additional libraries or software we plan to use free and open source. There are no costs associated with the project.

2. Scope

2.1. Subject area

We would like to create an editor that is easy to use and with objects that are highly modifiable, with focus on following areas:

- The games using this editor shall be able to expose any number of game objects by using special assets.
- The editor shall save world information in the format the engine already uses.
- The game can use a core set of objects and features so it can understand the noxed objects without including the entire level editor.

2.2. Limits/requirements.

The game(s) and editor should run in Windows 7 or newer and Ubuntu 14 LTS. The programs should start on integrated graphic cards, but need only run smoothly on newer dedicated graphic cards under, not accounting for exaggerated level designs with a high object density..

2.3. Assignment

We will make at least one game and a level editor to said game. If time permits we will make multiple games and make the editor work on all of them.

3. Project Organization

3.1. Responsibilities and roles

Team Leader: Tor Strømme

No other roles defined seeing as we are only two persons.

3.2. Routines and rules.

- Any decision made for the group should be discussed, if no agreement can be reached we will first discuss with an external (Magnus from Suttung or Simon from NTNU). If after a discussion no agreement is reached, the group leader will make a decision.
- If a member of the group has not performed his assigned tasks in the agreed upon time it is up to the other member to give a verbal warning. If the behaviour is repeated a meeting will be called with Simon-NTNU.
- Any unforeseen costs will be split evenly among the members.
- Any member has the right to sign and authorize on behalf of the group.
- When work hours are concerned our main objective each week will be to complete assigned tasks. However we expect each member to spend at least 20 hours each week even if the task is easy or completed early. The goal is to have an average around 30 hours.

4. Planning and reports:

4.1. Workflow

We will follow the Scrum model as much as is sensible with two persons. Mainly we will have four major parts: Design, develop game, develop game editor and completion. The game and the editor will be developed in tandem. We allocated 9 weeks starting week 6 to develop our software. The remaining time will be used for bug fixes, polishing and report writing.

We will start with weekly sprints working together in the same room, though this is subject to change as the project matures. If we do not physically meet on a work day we will have the daily scrum meeting on skype instead. Every monday at 13.00 we will hold the Scrim Review and Retrospective for the completed sprint followed by the Planning Meeting for the next sprint. Any member can call meetings and any member is expected to update the Backlog.

4.2. External Meetings

Meet Suttung/Magnus once a week, mondays 1400. Simon will be at the Mustad rooms we are using at Wednesdays so we will have meetings with him there approximately every other week.

5. Quality Assurance

5.1. Documentation

All commits to bitbucket should be complete with useful comments(see 5.3). Classes, functions and resources should be commented in a fashion that we can generate documentation using Doxygen or similar software.

5.2. Code Style and Quality

We will attempt to use the c++ core guidelines.

5.3. Configuration Management

Issues and tasks will be tracked using Jira. We chose Jira since it is popular and we figure that it is easier to get help if lots of people use it.

Git will be used for source control. Any errors with integration will be reported when discovered. The author of a class or function will have a responsibility that it works in the context of the software and should be available for figuring out what went wrong.

Every new commit to the Master Branch will be preceded by a code review with the other group member. This is to ensure that the latest version always is working and satisfies all and any standards.

When we get a working copy of the game and editor the newest versions will be made available on the NTNU bachelor project web site.

5.4. Risk Analysis

Event	Probability	Impact Assessment
Sickness/disease	Medium	Depends on duration and severity.

Absence/procrastination	Medium	High
Hardware Issues	Low	Low
Technical Issues with game engine	High	Low
Downtime on web services	Low	Low
Lack of Communication	Medium	High
Loss of Group Member	Low	High

Preemptive/reactive measures:

Disease/Sickness:

- Notify the other group member. Make sure any work you have done is documented and available.

Absence/Procrastination:

- Communicate what is happening on the group meetings. Motivate each other.

Hardware Issues:

- Make sure you save/commit your work regularly. We both have backup computers.

Technical issues with Game Engine:

- Contact Suttung and make sure to create a bug on their tracker. Try to work around it in the short term. If it is easily fixed, create a pull request.

Downtime on web services:

- Upload local copies to alternative sources and keep working. The services we use are using are very robust and reliable. We expect at most to have very short time issues.

Lack of Communication

- Weekly and daily meetings should be enough to counteract this.

Loss of Group Members

- Make sure everyone is working at an even pace and that motivation is kept high. If against all odds a member is excluded a meeting has to be called to find out how and what can be delivered.

6. Work Planning

6.1. Main Parts

We have some flexibility in the planning phase since we are using Scrum. We have agreed to certain milestones we want to reach at certain times. What tasks need to be completed for each milestone is yet to see.

1. Project Plan delivery: 28/01-16
2. Prototype game/editor: 22/02-16
    - A basic UI for the editor and a game with a player that can move around.
3. Main development complete: 10/04-16
    - The game and the editor is finished and complete. Only bug fixes and polish remains
4. Final Sprint: 09/05-16
    - Any and all development from here on out will be if the code does not compile and run on the target systems. Code unrelated to that has to be committed to be merged or discarded by this date.
5. Report Delivery: 18/05-16

6.2. Assignments

We have planned on using 9 weeks for the development of features, but leave the last 5 for the written report, bug fixing, polishing or catching up if a feature took more time than first anticipated. We will have 5 major milestones and use our sprints to reach these.

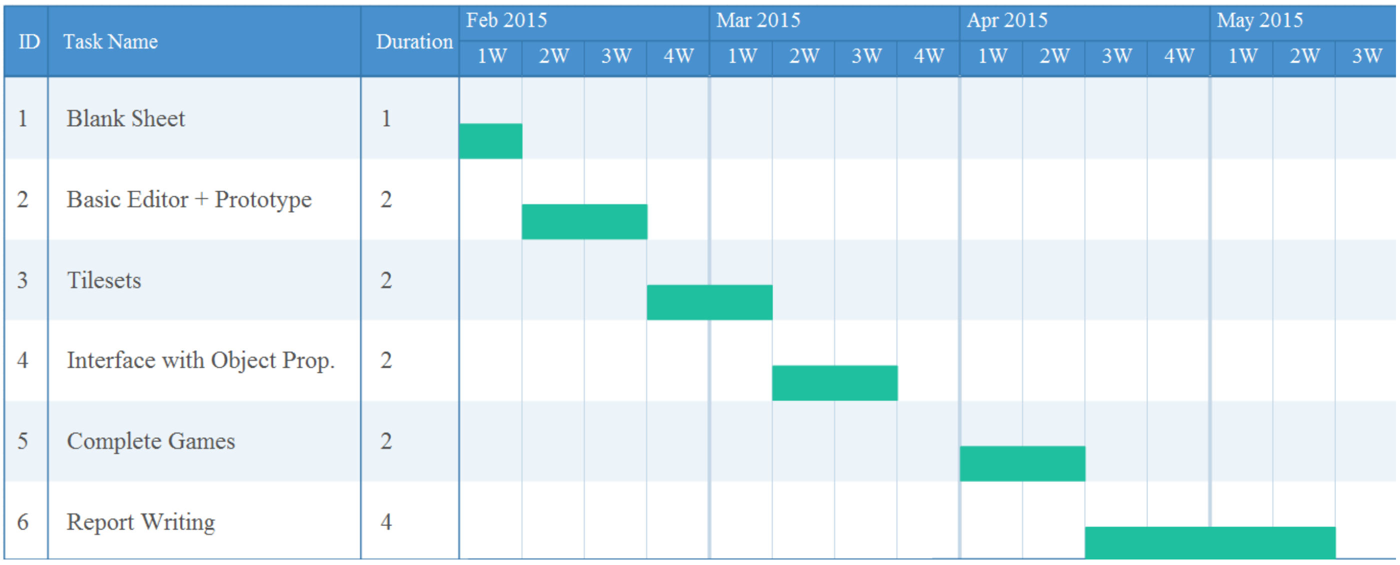
1. Blank Sheet to work from. 1 week.
2. Basic Editor with placement of assets. Simple Prototype of Game. 2 weeks.
3. Tilesets in both game and Editor. 2 weeks.
4. Interface in Editor with object properties. 2 weeks.
5. Complete Game(s). 2 weeks.
6. Report writing, bug fixes.

This is a list of features we would like to implement, in descending order of importance not order of implementation. If time is an issue we will discard from the bottom first.

- Editor: Entity placement
- Editor: Entity properties
- All: Tiled graphics for 2D level geometry
- Game: One example game with simple fighting, enemies, and movement.
- Editor: Tile set placement
- Editor: Different brushes
- All: Automatic adjacency-adapting tiles for faster tile map production.
- All: Meta-data for tiles that can be used by the game and editor.
- All: Integration of editor as submodule in projects.

- All: A separation of editor and core objects needed both by editor and game
- Game: Another example game with basic tower defence mechanics where a player can edit the map between attempts
- Game: Sound effects
- Game: Music

6.3. Gantt Chart



ID	Task Name	Duration	Feb 2015				Mar 2015				Apr 2015				May 2015		
			1W	2W	3W	4W	1W	2W	3W	4W	1W	2W	3W	4W	1W	2W	3W
1	Blank Sheet	1	🟩														
2	Basic Editor + Prototype	2		🟩	🟩												
3	Tilesets	2				🟩	🟩										
4	Interface with Object Prop.	2						🟩	🟩								
5	Complete Games	2									🟩	🟩					
6	Report Writing	4											🟩	🟩	🟩	🟩	

C Jira Worklog

[NOXED-76] TopDown: Find New Sprites for Player, PowerUp and Enemy.
Created: 07/mai/16 Updated: 07/mai/16

Status:	In Progress
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour		
Time Spent:	Not Specified		
Original Estimate:	1 hour		

Sprint:	Week 10
Story Points:	1

GUI: Key-Mapping Reference (use assets/control.json) (NOXED-48)

[NOXED-75] TopDown:Add sprites for Player, enemy and power-up Created: 07/mai/16 Updated: 07/mai/16 Resolved: 07/mai/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Technical task	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Unassigned
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 6, Week 7, Week 8

Comments

Comment by Tor Andreas Dyrlie Strømme [07/mai/16]

plangle is stupid

[NOXED-74] Refactor NoxedGame into Platformer and TopDown Created: 18/apr/16 Updated: 25/apr/16 Resolved: 25/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	2 hours		
Time Spent:	Not Specified		
Original Estimate:	2 hours		

Sprint:	Week 9
Story Points:	2

[NOXED-73] Platformer: Find and smarten tile set Created: 16/apr/16 Updated: 01/mai/16 Resolved: 01/mai/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 9, Week 10
Story Points:	3

Comments

Comment by Gisle Aune [01/mai/16]

The one I have will suffice

[NOXED-72] TopDown: Find and smarten tile set Created: 16/apr/16 Updated: 07/mai/16 Resolved: 07/mai/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Unassigned
Resolution:	Won't Fix	**Votes:**	0
Labels:	None		
Remaining Estimate:	3 hours		
Time Spent:	Not Specified		
Original Estimate:	3 hours		

Sprint:	Week 10
Story Points:	3

Comments

Comment by Tor Andreas Dyrlie Strømme [07/mai/16]

Will use the tileset already included in the editor. Will try to find some player and PowerUp/Enemy Assets

[NOXED-71] Platformer Add powerup Created: 16/apr/16 Updated: 01/mai/16 Resolved: 01/mai/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours, 5 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 10
Story Points:	3

[NOXED-70] TopDown: Add powerup Created: 16/apr/16 Updated: 07/mai/16 Resolved: 07/mai/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	6 hours, 12 minutes		
Original Estimate:	3 hours		

Sprint:	Week 9, Week 10
Story Points:	3

[NOXED-69] Platformer: Add enemy Created: 16/apr/16 Updated: 02/mai/16 Resolved: 02/mai/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	1 hour, 25 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 10
Story Points:	5

[NOXED-68] TopDown: Add enemy Created: 16/apr/16 Updated: 07/mai/16 Resolved: 07/mai/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	6 hours, 17 minutes		
Time Spent:	1 hour, 43 minutes		
Original Estimate:	1 day		

Sprint:	Week 10
Story Points:	8

[NOXED-67] Platformer: Add movement mechanics platformer Created: 16/apr/16 Updated: 20/apr/16 Resolved: 20/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	7 hours		
Original Estimate:	3 hours		

Sprint:	Week 9
Story Points:	3

[NOXED-66] TopDown: Add movement mechanics for topdown Created: 16/apr/16 Updated: 04/mai/16 Resolved: 03/mai/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours, 38 minutes		
Original Estimate:	3 hours		

Sprint:	Week 9, Week 10
Story Points:	3

[NOXED-65] Add Noxed-platformer and Noxed-topdown instead of noxed-game. Created: 16/apr/16 Updated: 25/apr/16 Resolved: 25/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Unassigned
Resolution:	Duplicate	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 9
Story Points:	2

[NOXED-64] Code Cleanup and Renaming of Gui-elements Created: 11/apr/16 Updated: 11/apr/16 Resolved: 11/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	1 hour		
Original Estimate:	1 hour		

Sprint:	Week 8
Story Points:	1

[NOXED-63] Make DearImhuiHandler handle more than one type of item at a time. Created: 10/apr/16 Updated: 11/apr/16 Resolved: 11/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	53 minutes		
Time Spent:	1 hour, 7 minutes		
Original Estimate:	2 hours		

Sprint:	Week 8
Story Points:	2

[NOXED-62] Move DearImguiHandler from DearImguiSubRenderer to NoxedEditorWindow Created: 10/apr/16 Updated: 11/apr/16 Resolved: 11/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Won't Fix	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour		
Time Spent:	Not Specified		
Original Estimate:	1 hour		

Sprint:	Week 8
Story Points:	1

Comments

Comment by Tor Andreas Dyrlie Strømme [11/apr/16]

GuiHandler needs to exist below DearImguiSubrenderer to render at the correct time.

[NOXED-61] GUI: Prevent click through Created: 06/apr/16 Updated: 09/apr/16 Resolved: 09/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 25 minutes		
Time Spent:	1 hour, 35 minutes		
Original Estimate:	3 hours		

Sprint:	Week 8
Story Points:	3

[NOXED-60] Make RandomTile work under Windows Created: 01/apr/16 Updated: 02/apr/16 Resolved: 02/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	1 hour		
Original Estimate:	Not Specified		

Sprint:	Week 7

[NOXED-59] GUI: Separate data structure for managing GUI elements, listening to instructions from components, and informing components that their GUI elements have had the value changed Created: 16/mar/16 Updated: 10/apr/16 Resolved: 10/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	1 day, 2 hours, 42 minutes		
Original Estimate:	8 minutes		

Sprint:	Week 6, Week 7, Week 8
Story Points:	8

Comments

Comment by Tor Andreas Dyrlie Strømme [09/apr/16]

added inputlayot and type to handler. #

[NOXED-58] GUI: Tidy up GUI code Created: 16/mar/16 Updated: 06/apr/16 Resolved: 06/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours, 30 minutes		
Original Estimate:	3 minutes		

Sprint:	Week 6, Week 7, Week 8
Story Points:	3

Comments

Comment by Tor Andreas Dyrlie Strømme [05/apr/16]

fixed imgui CMakestuff.

Comment by Tor Andreas Dyrlie Strømme [05/apr/16]

added GUI subrenderer to the game.

Comment by Tor Andreas Dyrlie Strømme [06/apr/16]

Testing VI

[NOXED-57] TileMap: "bigtile" Smart Tile Type (Multiple tiles in a repeating pattern, forming a big tile) Created: 16/mar/16 Updated: 16/mar/16 Resolved: 16/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	25 minutes		
Time Spent:	35 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

[NOXED-56] TileMap: "random" Smart Tile Type (has to be deterministic so that it renders exactly alike) Created: 16/mar/16 Updated: 16/mar/16 Resolved: 16/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	5 minutes		
Time Spent:	55 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

[NOXED-55] TileMap: "4bit" Smart Tile Type (example at http://www.saltgames.com/article/awareTiles/) Created: 16/mar/16 Updated: 21/apr/16 Resolved: 21/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	2 hours, 55 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6, Week 9
Story Points:	1

[NOXED-54] TileMap: Sometimes tiles do not resolve their neighbors right when placed quickly Created: 14/mar/16 Updated: 14/mar/16 Resolved: 14/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	30 minutes		
Time Spent:	30 minutes		
Original Estimate:	1 hour		

Story Points:	1

[NOXED-53] TileMap: "Island" smart tile that islands with others having a shared tag. Created: 13/mar/16 Updated: 13/mar/16 Resolved: 13/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Trivial
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	35 minutes		
Time Spent:	25 minutes		
Original Estimate:	1 hour		

Story Points:	0

[NOXED-52] WASD on the camera should send mouse move actions Created: 12/mar/16 Updated: 13/mar/16 Resolved: 13/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Unassigned
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	50 minutes		
Time Spent:	10 minutes		
Original Estimate:	1 hour		

Story Points:	1

Comments

Comment by Gisle Aune [13/mar/16]

This was a gross overestimation. I thought major rework had to be done, but it was just an update in application checking if the camera position matches the last known one. I needed to make a public method for broadcasting the mouse position. It takes the screen position and handles all the rest of the mouse vent.

[NOXED-51] "pushable" on the camera does nothing Created: 12/mar/16 Updated: 13/mar/16 Resolved: 13/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	12 minutes		
Time Spent:	3 minutes		
Original Estimate:	15 minutes		

Story Points:	0

[NOXED-50] Handler: State only considers last added keybinding's limit Created: 12/mar/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	10 minutes		
Time Spent:	5 minutes		
Original Estimate:	15 minutes		

Story Points:	0

[NOXED-49] Handler: MODE_SET also needs to affect state. Created: 12/mar/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	15 minutes		
Original Estimate:	10 minutes		

Story Points:	1

[NOXED-48] GUI: Key-Mapping Reference (use assets/control.json) Created: 12/mar/16 Updated: 25/apr/16 Resolved: 12/mar/16

Status:	Open
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Unassigned
Resolution:	Unresolved	**Votes:**	0
Labels:	None		
Σ Remaining Estimate:	5 hours	**Remaining Estimate:**	5 hours
Σ Time Spent:	Not Specified	**Time Spent:**	Not Specified
Σ Original Estimate:	1 day	**Original Estimate:**	1 day

Sub-Tasks:

Key	Summary	Type	Status	Assignee
NOXED-75	TopDown :Add sprites for Player, enemy...	Technical task	Closed	

Sprint:	Week 6, Week 7, Week 8
Story Points:	8

[NOXED-47] GUI: Entity Managment (Get/Set Property) Created: 12/mar/16 Updated: 12/apr/16 Resolved: 12/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 7, Week 8
Story Points:	3

[NOXED-46] GUI: Communication Event that can describe content and visibily of a named block of GUI elements, e.g. entity properties. Created: 12/mar/16 Updated: 01/apr/16 Resolved: 01/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 7
Story Points:	3

[NOXED-45] GUI: TileMap Managment (Add/Remove, List) Created: 12/mar/16 Updated: 15/apr/16 Resolved: 15/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	1 day, 30 minutes		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 7, Week 8
Story Points:	3

Comments

Comment by Gisle Aune [12/apr/16]

Mistyped issue number. This one is not done!

[NOXED-44] GUI: Cursor Information (Position, Mode, [Size]) Created: 12/mar/16 Updated: 12/apr/16 Resolved: 12/apr/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	10 minutes		
Time Spent:	1 hour, 50 minutes		
Original Estimate:	2 hours		

Sprint:	Week 6, Week 7, Week 8
Story Points:	2

[NOXED-43] TileMap: Make it possible for code to look for a specific smart tile tag and query the smart tile about entity creation parameters on world save Created: 12/mar/16 Updated: 13/mar/16 Resolved: 13/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 50 minutes		
Time Spent:	3 hours, 10 minutes		
Original Estimate:	5 hours		

Story Points:	5

Description

For the early rough tilemapping, so that the designer need not place individual tiles.

Comments

Comment by Gisle Aune [13/mar/16]

Went almost 2 hours below the estimate

[NOXED-42] TileMap: Make brush size for rapid rough tile placement, and 2 brush shapes (diamond, square) Created: 12/mar/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	30 minutes		
Time Spent:	4 hours, 30 minutes		
Original Estimate:	5 hours		

Story Points:	5

Comments

Comment by Gisle Aune [12/mar/16]

Upped the estimate because this requires a new renderer too.

[NOXED-41] Add GUI-rendering to the DearImguiSubrenderer. Created: 10/mar/16 Updated: 16/mar/16 Resolved: 16/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	3 hours		
Time Spent:	Not Specified		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 6
Story Points:	3

Description

Add the actual render functions to the DearImguiSubrender class.

Comments

Comment by Tor Andreas Dyrlie Strømme [15/mar/16]

imgui added as submodule

Comment by Tor Andreas Dyrlie Strømme [16/mar/16]

Added ugly version. Will be cleaned up in issue NOXED-59

Comment by Tor Andreas Dyrlie Strømme [16/mar/16]

*NOXED-58

[NOXED-40] Add classes NoxedEditorApplication and NoxedEditorView for Editor specific stuff. Created: 10/mar/16 Updated: 14/mar/16 Resolved: 14/mar/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour		
Time Spent:	4 hours		
Original Estimate:	5 hours		

Sprint:	Week 6
Story Points:	5

Description

If we are to use a nox::subrender for rendering the gui on top of the editor we will need to create a new class that extends the NoxedApplication class so that we can also extend the NoxedWindowView class to use the renderer.

[NOXED-39] Create Subrenderer for GUI rendererring Created: 09/mar/16 Updated: 10/mar/16 Resolved: 10/mar/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours, 25 minutes		
Original Estimate:	2 hours		

Sprint:	Week 6
Story Points:	2

Description

Add a subrenderer object that can be hooked onto the renderer. Need to override RenderSdlWindowView::OnWindowCreated() to add it when program starts.

Comments

Comment by Tor Andreas Dyrlie Strømme [10/mar/16]

During this issue I discovered that we need to refactor a bit, will create new issues.

[NOXED-38] Make ModeChange stateful with toggles, addition and subtraction
Created: 08/mar/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 25 minutes		
Time Spent:	35 minutes		
Original Estimate:	2 hours		

Story Points:	2

[NOXED-37] Make tags for smart tiles handled in base class Created: 08/mar/16 Updated: 11/mar/16 Resolved: 11/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	50 minutes		
Time Spent:	10 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

[NOXED-36] Hold down mouse button to place multiple tiles. Created: 07/mar/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	40 minutes		
Time Spent:	1 hour, 20 minutes		
Original Estimate:	2 hours		

Sprint:	Week 6
Story Points:	2

[NOXED-35] Make it possible to "dumb down" the smart tile set when saving the world, to make the game use less resources with a tilesat that is unchanging anyway
Created: 07/mar/16 Updated: 11/mar/16 Resolved: 11/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	50 minutes		
Time Spent:	10 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

[NOXED-34] Make the tile map saved with the world show up in the game. Created: 07/mar/16 Updated: 11/mar/16 Resolved: 11/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 40 minutes		
Time Spent:	20 minutes		
Original Estimate:	2 hours		

Sprint:	Week 6
Story Points:	2

Comments

Comment by Gisle Aune [10/mar/16]

Changed world saver component. See relevant commit

Comment by Gisle Aune [11/mar/16]

Just making it assigned since I forgot to in progress it

[NOXED-33] Serialize TileMap component Created: 07/mar/16 Updated: 10/mar/16 Resolved: 10/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	45 minutes		
Time Spent:	15 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

[NOXED-32] Use a deletable flag on EditorBlueprint, and check for it on occupancy checks. If it's not there, the occupant should be ignored. Created: 05/mar/16 Updated: 11/mar/16 Resolved: 11/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 49 minutes		
Time Spent:	11 minutes		
Original Estimate:	2 hours		

Sprint:	Week 6
Story Points:	2

[NOXED-31] Serialize TileGrid (see the deserialize method for structure) Created: 05/mar/16 Updated: 10/mar/16 Resolved: 10/mar/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	45 minutes		
Time Spent:	15 minutes		
Original Estimate:	1 hour		

Sprint:	Week 6
Story Points:	1

Comments

Comment by Gisle Aune [10/mar/16]

Will merge with master after tilemap serialize issue

[NOXED-30] Fix off-by-one error in EntitySpawner causing a segfault when spawning entity with id = length of spawnable actor list Created: 05/mar/16 Updated: 05/mar/16 Resolved: 05/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Bug	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Unassigned
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	59 minutes		
Time Spent:	1 minute		
Original Estimate:	1 hour		

Story Points:	1

Comments

Comment by Gisle Aune [05/mar/16]

Swept that one up while working on issue 28

[NOXED-29] Make a Tiler component, and a way of receiving its instructions to edit a tileset Created: 05/mar/16 Updated: 05/mar/16 Resolved: 05/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour		
Time Spent:	2 hours		
Original Estimate:	3 hours		

Sprint:	Week 5
Story Points:	3

Comments

Comment by Gisle Aune [05/mar/16]

And the tiler is done. Now all that remains is getting smart tileset working, and the schedule is preserved

[NOXED-28] Make a mode change for the grid cursor object so that different placer components can cooperate. Created: 05/mar/16 Updated: 05/mar/16 Resolved: 05/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	40 minutes		
Time Spent:	20 minutes		
Original Estimate:	1 hour		

Sprint:	Week 5
Story Points:	1

Comments

Comment by Gisle Aune [05/mar/16]

ModeChange event

Comment by Gisle Aune [05/mar/16]

That didn't take long. Use E and T for the two current modes, as defined in assets/control.json

[NOXED-27] Doxygen comment cleanup Gisle Created: 02/mar/16 Updated: 09/mar/16 Resolved: 09/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	40 minutes		
Time Spent:	2 hours, 20 minutes		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 5
Story Points:	3

[NOXED-26] Doxygen comment cleanup Tor Created: 02/mar/16 Updated: 13/apr/16 Resolved: 13/apr/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Task	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour, 30 minutes		
Time Spent:	1 hour, 30 minutes		
Original Estimate:	3 hours		

Sprint:	Week 6, Week 5, Week 6, Week 7, Week 8
Story Points:	3

[NOXED-25] Delete/Disable collided object. Created: 29/feb/16 Updated: 02/mar/16 Resolved: 02/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	1 hour		
Time Spent:	4 hours		
Original Estimate:	5 hours		

Sprint:	Week 4
Story Points:	5

[NOXED-24] Make the smart tile set with 3 basic types of smart tiles, and make the tile map able to use it Created: 25/feb/16 Updated: 07/mar/16 Resolved: 07/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	5 hours, 5 minutes		
Original Estimate:	5 hours		

Sprint:	Week 4, Week 5
Story Points:	5

Comments

Comment by Gisle Aune [06/mar/16]

Smart tiles are loaded now, and there's one type which is not very smart.

Comment by Gisle Aune [06/mar/16]

The tilemap works now with smart tiles, but there's just one boring type to choose from and it's not resolving neighbors yet. Overtime is very possible, but 2 more tile types and neighbor resolution is all that remains.

Comment by Gisle Aune [07/mar/16]

Finished it up to what the issue said. The tile set has 6 smart tiles, check them out with keys 1-6 in the editor after setting editor mode with 'T'-key. The three types to demonstrate it is: static (just plain old one-to-one), chequered (chess board), and island (inner and outer layer)

[NOXED-23] Make tilemap component that uses tileset resource Created: 25/feb/16 Updated: 05/mar/16 Resolved: 05/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	2 days, 50 minutes		
Original Estimate:	5 hours		

Sprint:	Week 4, Week 5
Story Points:	5

Comments

Comment by Gisle Aune [28/feb/16]

Made TileGrid class to organize them, but it's untested.

Comment by Gisle Aune [28/feb/16]

Fixed the TileGrid class and made the boilerplate code for the TileMap

Comment by Gisle Aune [03/mar/16]

There's a bad alloc error. Commiting to check it in VS

Comment by Gisle Aune [04/mar/16]

This has taken longer than expected, but there's some tile action going on now! 😃 I need some more optimizations and cleanups, and then it should be ready for closing

Comment by Gisle Aune [05/mar/16]

The tile map is complete, however it is still not editable as that is beyond the scope of this issue. This was a gross underestimation!

[NOXED-22] Make tileset resource that loads on startup. Created: 25/feb/16 Updated: 28/feb/16 Resolved: 28/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours, 25 minutes		
Original Estimate:	3 minutes		

Sprint:	Week 4
Story Points:	3

Comments

Comment by Gisle Aune [27/feb/16]

Tile Set resource is working. Nothing visible yet, however.

[NOXED-21] Test imgui with Noxed Created: 24/feb/16 Updated: 09/mar/16 Resolved: 09/mar/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	7 hours		
Original Estimate:	5 hours		

Sprint:	Week 6, Week 5
Story Points:	5

Comments

Comment by Tor Andreas Dyrlie Strømme [09/mar/16]

Imgui will work the engine, but the creation of a subrenderer is necessary.

[NOXED-20] Add Collision Detection on Player Object. Created: 24/feb/16 Updated: 24/feb/16 Resolved: 24/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours		
Original Estimate:	3 hours		

Sprint:	Week 4
Story Points:	3

[NOXED-19] Be able to spawn two different entities in editor. Created: 22/feb/16 Updated: 25/feb/16 Resolved: 25/feb/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	2 hours, 40 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3, Week 4

Comments

Comment by Gisle Aune [22/feb/16]

Started working on keyboard mode change event and handler.

Comment by Gisle Aune [23/feb/16]

Made the mode changer now, which should pave the way to making this possible before we get a GUI. See EntitySpawner component for test code.

Comment by Gisle Aune [25/feb/16]

The editor can now select actors to spawn, see the editor's actor/Cursor.json and controls.json resource for how that's bound for now.

[NOXED-18] Split spawner code out of GridCursor Created: 22/feb/16 Updated: 22/feb/16 Resolved: 22/feb/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	20 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

Comments

Comment by Gisle Aune [22/feb/16]

EntitySpawner handles the entity spawning GridCUrsor did prior. Merging it into master now.

[NOXED-15] Remove code from Window View for old mouse event. (Requires: NOXED-12, NOXED-13, NOXED-7) Created: 16/feb/16 Updated: 12/mar/16 Resolved: 12/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	50 minutes		
Time Spent:	10 minutes		
Original Estimate:	1 hour		

Story Points:	1

[NOXED-14] Place exposed entities (Requires: NOXED-10) Created: 15/feb/16 Updated: 20/feb/16 Resolved: 20/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Done	**Votes:**	0
Labels:	Editor		
Remaining Estimate:	0 minutes		
Time Spent:	5 hours, 10 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

Comments

Comment by Gisle Aune [20/feb/16]

Started work, but got segfault when saving.

Comment by Gisle Aune [20/feb/16]

Tried fixing some errors.

Comment by Gisle Aune [20/feb/16]

The culprit was a lot of & in return types. It almost works now, except the components in the blueprint

Comment by Gisle Aune [20/feb/16]

Exporting works now. Fixed json error in actor Wall on editor. Added transform.

Comment by Gisle Aune [20/feb/16]

World saving now saves them correctly. Fixed "extends" typo that caused save objects to not extend

Comment by Gisle Aune [20/feb/16]

Deleting works again, but we may need to find another way to detect grid occupancy.

Comment by Gisle Aune [20/feb/16]

We worked together on this issue, hence the doubled time.

[NOXED-13] Make Camera moving component use new mouse event (Requires: NOXED-7) Created: 15/feb/16 Updated: 17/feb/16 Resolved: 17/feb/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	Editor		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 3

Comments

Comment by Gisle Aune [17/feb/16]

This did not become a problem.

[NOXED-12] Make GridCursor use new mouse event (Requires: NOXED-7)
Created: 15/feb/16 Updated: 16/feb/16 Resolved: 16/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	20 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

[NOXED-11] Hotkey event for ctrl, alt or shift-modified key-presses Created: 15/feb/16 Updated: 14/mar/16 Resolved: 14/mar/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Minor
Reporter:	Gisle Aune	**Assignee:**	Unassigned
Resolution:	Fixed	**Votes:**	0
Labels:	Noxed		
Remaining Estimate:	20 minutes		
Time Spent:	1 hour, 40 minutes		
Original Estimate:	3 hours		

Story Points:	2

Comments

Comment by Gisle Aune [05/mar/16]

Can be added as extra functionality in the ModeChange event emitter in handlers/

[NOXED-10] Expose to Editor Created: 15/feb/16 Updated: 20/feb/16 Resolved: 20/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	Editor		
Remaining Estimate:	0 minutes		
Time Spent:	1 day, 1 hour, 10 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

Description

Assets for exposing game entities for the level editor to place and adjust.

Comments

Comment by Gisle Aune [19/feb/16]

Code does not compile and there's nothing to demo yet.

Comment by Gisle Aune [19/feb/16]

Added files to cmake list

Comment by Gisle Aune [20/feb/16]

Class set up, but not confirmed working. It compiles, at least.

Comment by Gisle Aune [20/feb/16]

This should work, but I need a new branch to weave this into the save-the-world stuff before I know for sure.

Comment by Gisle Aune [20/feb/16]

just going to add time

Comment by Gisle Aune [20/feb/16]

Half of those hours was wasted doing some crazy solution, though.

[NOXED-9] Cursor Camera that moves with WASD and when mouse is near edge
Created: 15/feb/16 Updated: 17/feb/16 Resolved: 17/feb/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Done	**Votes:**	0
Labels:	Editor		
Remaining Estimate:	0 minutes		
Time Spent:	5 hours, 20 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

Comments

Comment by Gisle Aune [17/feb/16]

Tue 16:30-16:45
Wed 07:45-11:20, 12:45-14:15
Total: 5h 20m

The camera components are for controlling the view's main camera. If need arises, some refactoring would be necessary to have it only control a local camera, which it will have to acquire through a locally broadcasted component event.

The first thing I did was allowing other objects access to the camera from the view. However, getting the view proved to be a challenge. Finding the controlled view returned null despite the actor being controlled.

After more investigation, I found out that the function to find the controlling view explicitly finds the view if the view is NOT controlling the actor. Changing the operator from != to == fixed the issue I had, but I contacted the engine devs about this and asked if it was the intended behavior.

Logic.cpp:126: if (controlledActor && controlledActor->getId() != actorId)

Magnus asked that I send a pull request of the change, and so I did. The pull request ended up rebased into commit 2f07c2d319dbd805ed7b1d3d1736b7aa3a62960e. That solves that problem, and I finished the class for the follow-camera.

Next task was the cursor camera, which shall only move when the cursor is close to the edge or the keys. Apparently I can use the camera's BBOX for this, making the MouseAction properties I made with this in mind moot.

WASD movement was a minor challenge, but I needed to figure out that the keyboard event was press and release. I added a keyboardMovement vector for that to be added. One issue with the WASD movement is the grid cursor not moving unless there's been a change. That's out of scope for this issue, but it will have to be addressed as its own little issue.

The reason I jumped to that was that it showed me the key to a problem with the mousemovement. It would accelerate when a constant speed was intended. I changed it to using a movement vector, as well, instead of trying to have it move towards a target position. Normalization if both was at zero made it fail completely, but adding a small threshold before new position was calculated fixed it.

[NOXED-8] Save the world! Created: 15/feb/16 Updated: 20/feb/16 Resolved: 20/feb/16

Status:	Closed
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	Editor		
Remaining Estimate:	0 minutes		
Time Spent:	4 hours		
Original Estimate:	Not Specified		

Sprint:	Week 3

Comments

Comment by Tor Andreas Dyrlie Strømme [20/feb/16]

Merged with master

[NOXED-7] Class for mouse event Created: 15/feb/16 Updated: 16/feb/16 Resolved: 16/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Gisle Aune	**Assignee:**	Gisle Aune
Resolution:	Fixed	**Votes:**	0
Labels:	Noxed		
Remaining Estimate:	0 minutes		
Time Spent:	1 hour, 55 minutes		
Original Estimate:	Not Specified		

Sprint:	Week 3

Description

- ❒ class MouseAction : Event
- ❒ Event needs the following methods and correspoing variables
- ❒ bool isClick()
- ❒ int getMouseButton()
- ❒ glm::ivec2 getScreenPosition()
- ❒ glm::vec2 getWorldPosition()
- ❒ Use constants for button names left, right and middle
- ❒ Integrate it into NoxedWindowView class

Comments

Comment by Gisle Aune [16/feb/16]

There are most likely a number of entites that wants to listen to mouse clicks, but not movements. Add that to keyboard events not needing be fired because the cursor moved, and the call for a new event class is there. The biggest motivator, however, is that the Action event doesn't have enough space for all the variables that mouse events need.

The solution is two event classes, MouseMoveEvent and MouseClickEvent. The latter inherits from the former. That will mean there are Actors that use both, but those that only need to listen to clicks won't have to check an event every frame.

Or so I thought. When doing that, the event ID would be the same for both. After looking into the event type names, I found out that I could merge those two classes into one and have two names for it. MouseAction::MOVE and MouseAction::BUTTON. They are both used like Action::ID for listening purposes. This may be a premature optimization, but the thing I learned about events should be useful down the line.

[NOXED-5] Second type of environment, power-up/destroyable, interact with the player in some way. Created: 15/feb/16 Updated: 24/feb/16 Resolved: 24/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Fixed	**Votes:**	0
Labels:	Game		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 3, Week 4

Comments

Comment by Tor Andreas Dyrlie Strømme [22/feb/16]

Added a game application class for game specific stuff, also made a second actor json for testing.

Comment by Tor Andreas Dyrlie Strømme [24/feb/16]

Splitting up in smaller issues.

[NOXED-3] Improve Codebase structure: Noxed, Noxed-game and Noxed-editor Created: 09/feb/16 Updated: 15/feb/16 Resolved: 15/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	Improvement	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 2

Comments

Comment by Gisle Aune [15/feb/16]

Projects use the code from Noxed, by including it into the projects. The project name needs to be supplied to the NoxedApplication class.

[NOXED-2] Editor: Place wall "clumps" Created: 09/feb/16 Updated: 15/feb/16 Resolved: 15/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Gisle Aune
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	Not Specified		
Time Spent:	Not Specified		
Original Estimate:	Not Specified		

Sprint:	Week 2

Comments

Comment by Gisle Aune [15/feb/16]

Clicking the editor places walls. This is test code to be replaced with a proper level editor placement.

[NOXED-1] Game: Collision with walls Created: 09/feb/16 Updated: 09/feb/16 Resolved: 09/feb/16

Status:	Resolved
Project:	2016: Noxed
Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

Type:	New Feature	**Priority:**	Major
Reporter:	Tor Andreas Dyrlie Strømme	**Assignee:**	Tor Andreas Dyrlie Strømme
Resolution:	Done	**Votes:**	0
Labels:	None		
Remaining Estimate:	0 minutes		
Time Spent:	3 hours		
Original Estimate:	Not Specified		

Sprint:	Week 2

Comments

Comment by Tor Andreas Dyrlie Strømme [09/feb/16]

There now are wall clump objects that you can collide with in the noxed-game