**WonderMaize**

5/11/16

Team D

Matthew Persing

Alec Tiefenthal

Jack Porter

Contents

[Introduction 3](#_Toc450720354)

[WonderMaize User Guide 4](#_Toc450720355)

[Objective 4](#_Toc450720356)

[Basic Controls 4](#_Toc450720357)

[Gameplay 4](#_Toc450720358)

[System Requirements 6](#_Toc450720359)

[Operating System Compatibility 6](#_Toc450720360)

[Hardware Compatibility 6](#_Toc450720361)

[Installation Instructions 6](#_Toc450720362)

[Maintenance Guide 7](#_Toc450720363)

[Tools 7](#_Toc450720364)

[Troubleshooting 7](#_Toc450720365)

[Software Requirements Specification (SRS) 8](#_Toc450720366)

[Software and Architecture Design Specification (SADS) 9](#_Toc450720367)

[Test Plan/Strategy 10](#_Toc450720368)

# Introduction

WonderMaize is a maze game for maze enthusiasts. Great for all ages so long as the user thinks they can navigate a maze. Within this documents are an in depth user guide that explains how to start and play WonderMaize as well as an installation and maintenance guide.

# WonderMaize User Guide

## Objective

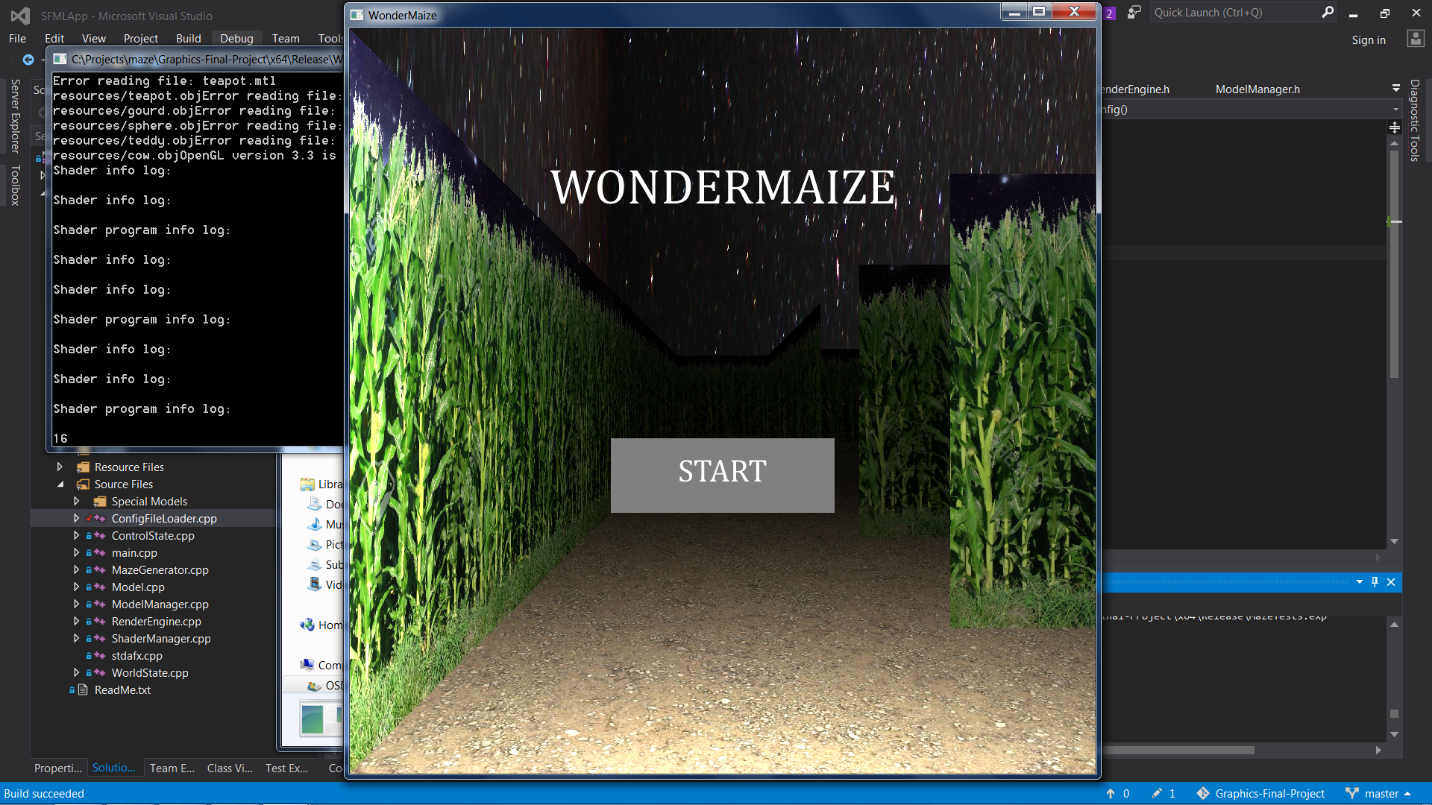
The objective of the WonderMaize game is to navigate through the randomly generated maze to find 5 different mysterious objects hidden.

## Basic Controls

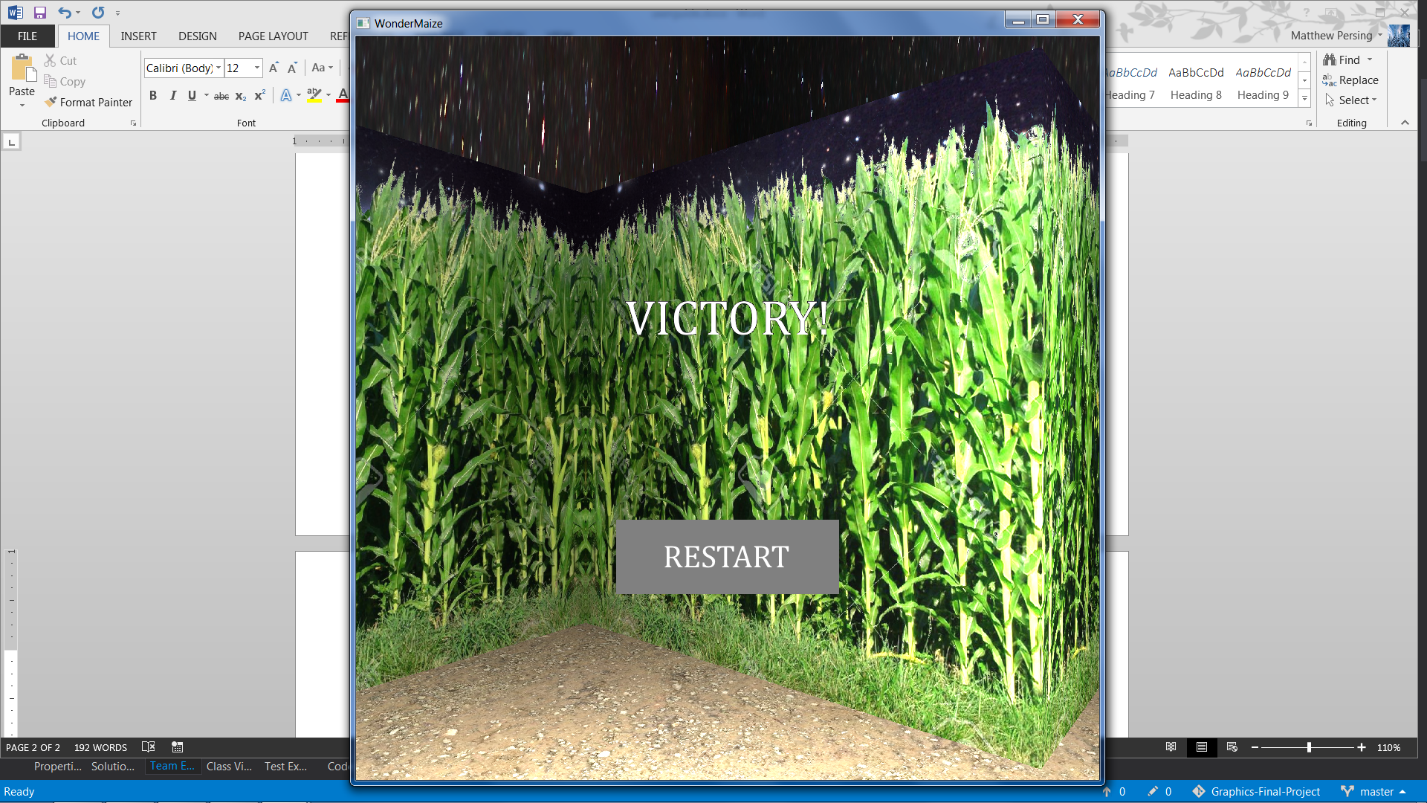
|  |  |
| --- | --- |
| Key | Action |
| W | Move forward |
| A | Turn left |
| S | Move backwards |
| D | Turn right |
| Esc | Quit the game |

## Gameplay

Upon opening the game, a box will pop up to begin playing the game. Click in the box to begin playing the game.

Once in the maze, the goal is to find all 5 hidden objects. The objects that are hidden in the maze are a Cow, Gourd, Sphere, Teapot, and Teddy Bear. Initially all objects are colored red. To actually find the object, simply run close to it and a swirly effect will appear on the screen and the object will change color to yellow.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Sphere | Teapot | Gourd | Teddy Bear | Cow |

After finding all of the models, a screen showing that the player has won will appear. To restart the game, press the “Restart” button in the middle of the screen. After pressing the “Restart” button, a new maze will be randomly generated and the objects to find will be randomly scattered throughout the maze.

# System Requirements

## Operating System Compatibility

|  |  |
| --- | --- |
| Windows 7 (32-bit) | Not Supported |
| Windows 7 (64-bit) | Supported |
| Windows 8 (32-bit) | Not Supported |
| Windows 8 (64-bt) | Not Supported |
| Windows 10 (32-bit) | Not supported |
| Windows 10 (64-bit) | Untested |
| Mac OSX | Not supported |
| Linux | Not supported |

## Hardware Compatibility

A graphics card with OpenGL 3.3 or newer is required for proper operation.

## Installation Instructions

1. Unzip the provided .zip file into a new directory.
2. Run the vc\_redist.x64.exe installer to install the Visual Studio 2015 Runtime. This is required to run the WonderMaize.exe
3. Run WonderMaize.exe to open the game. The previous steps do not need to be repeated in the future.

# Maintenance Guide

Tools

* Microsoft Visual Studio (2013 or newer)
* OpenGL
* Glew
* GitLab
* SFML
* Blender

Troubleshooting

|  |  |
| --- | --- |
| Program not start when launched | If this fails try reinstalling and making sure all files are in the appropriate folders. |
| Program does not run | Check software compatibility to ensure the software will run on your system. Ensure files are as they were after the initial installation. |
| Receive unable to open file error. | Ensure the config.txt file is located in the directory that the game is located in. For the release build look in the x64/Release folder. If attempting to run from Visual Studio look in the WunderMaze folder. |

# Software Requirements Specification (SRS)

The application should present the user with a randomized 3D maze. The controls for the game should be intuitive, requiring little or no instruction to users who have played any type of 3D game on a computer before. The maze should have some kind of end condition, like finding various objects hidden within it. The maze should be easily repeatable and customizable, with the customization made possible with at least some kind of configuration file. The program should not encounter any errors caused on its own accord. In the event that there is an error, fatal or otherwise, due to an issue outside the scope of the application, the error should *never* affect the running of a new instance of the application (if the application is closed and reopened, it should work regardless of what happened in the previous session).

# Software and Architecture Design Specification (SADS)

# Test Plan/Strategy

The WonderMaize game’s initial manifestation was for demonstrating Computer Graphics, and therefore a large percentage of the code is for front end graphics calculations and execution. To reduce potential errors in the WonderMaize, a test harness has been created in MSTest to test the backend code for the project. The front end code consists of calls to SFML and OpenGL and therefore is not practical to test. The backend classes that are tested in the test harness are MazeGenerator, ControlState, and objLoader. These are the primary classes that don’t have interaction with OpenGL and are therefore able to be tested decently well.

The MazeGenerator tests are largely characterization tests that test the properties of the MazeGenerator. The tests included for MazeGenerator are:

* TestInitialization: Basic test to make sure the MazeGenerator loads.
* TestMazeSizeGetters: Make sure that the getXSize() and getYSize() methods reflect the arguments that makeMaze was called with.
* TestMazeBorders: Make sure that the maze has all outer walls created in a simple square maze.
* TestNonsquareMazeBorders: Make sure that the maze has all outer walls created in a rectangular maze.
* TestMinMazeSize: Boundary value. Make sure that the maze properly generates at the smallest allowed size.
* TestMazeXSizeTooSmall: Boundary value. Make sure that the maze is not created given too small of an X dimension.
* TestMazeYSizeTooSmall: Boundary value. Make sure that the maze is not created given too small of a Y dimension.

The ControlState tests test the user input side of the games state. The tests included for ControlState are:

* TestInitializeation: Basic test to make sure ControlState loads
* TestStep: Very simple test to check that basic movement within a cell updates state correctly.
* TestStepWithTurning: A more exhaustive test that checks that steps update information correctly after repeated sequences of steps and turns.
* TestCellLocation: An exhaustive test that checks to see that the state handles exploring into cells in 3D space correctly and accurately.