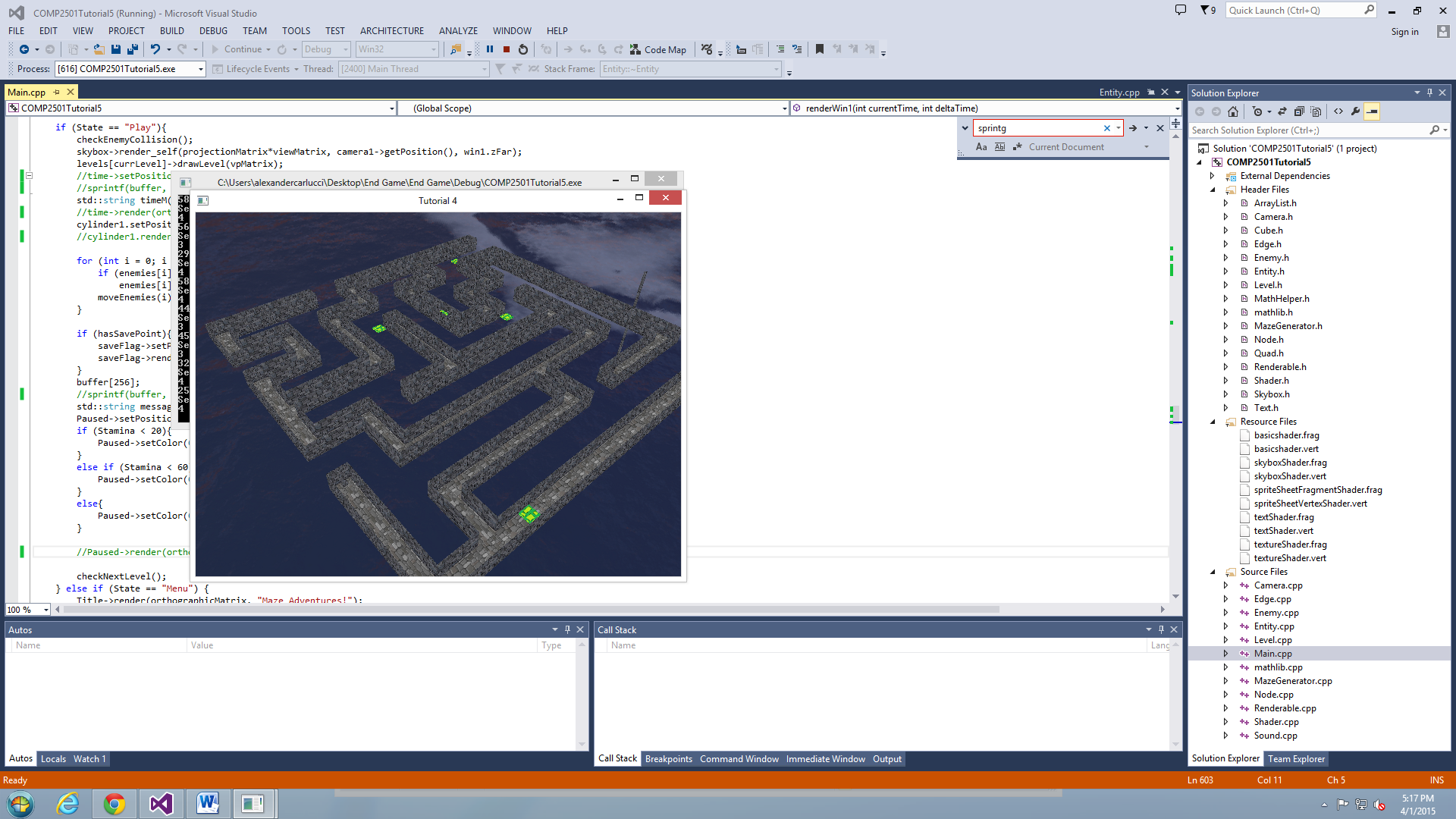
**Maze Adventures**

**Joel Scarfone 100947766**

**Alex Carlucci 100923882**



**User Guide**

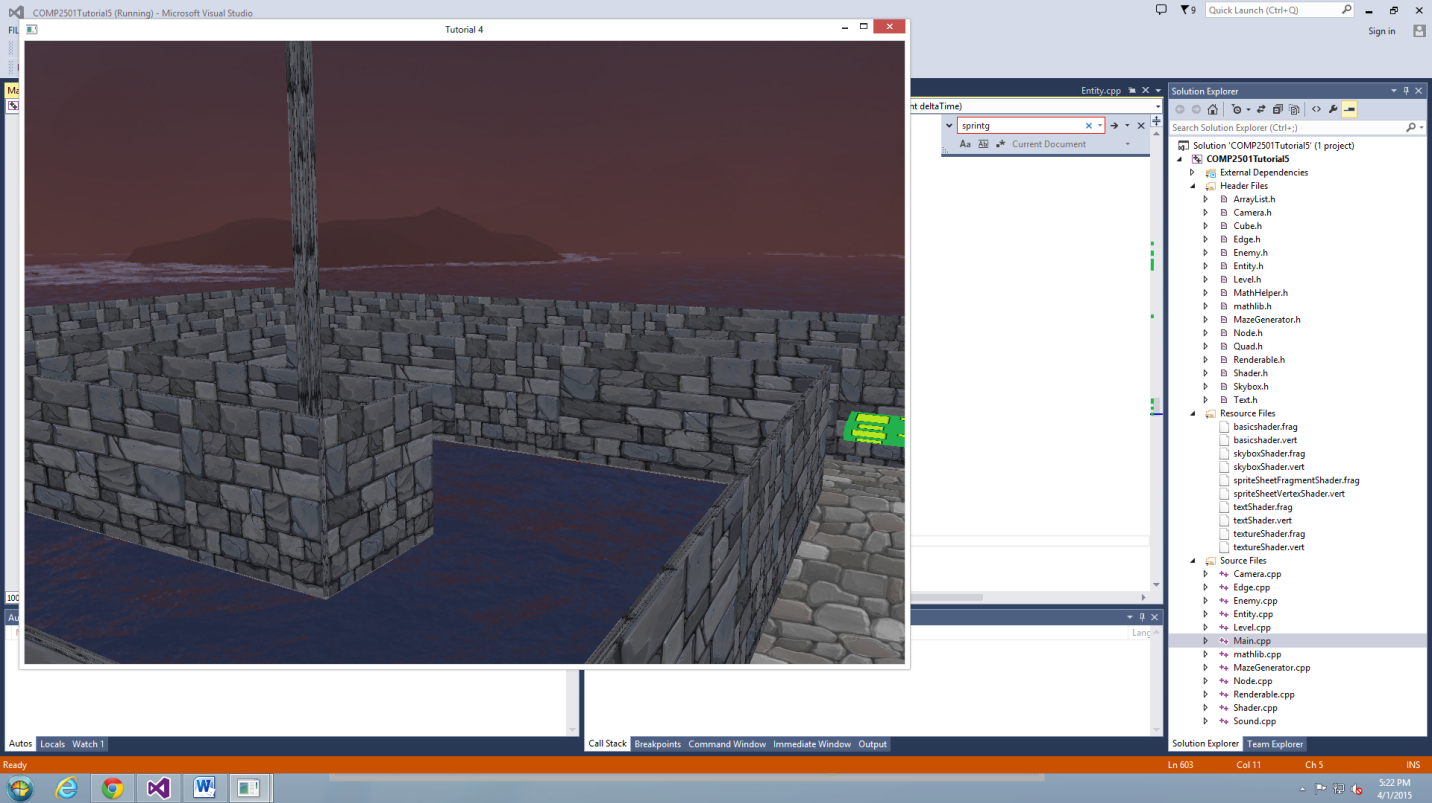
Maze Adventures is takes place in a ‘dream world’. The player is first emerged into the world where he can freely look around and walk where he wishes. Very quickly, the player will discover that they are stuck in a maze. They will notice a pillar somewhere in the maze that stands out, and must get there to solve the maze. On arrival to the pillar (end of the maze), the player will have completed that level of the dream. Each level gets progressively harder until the user completes level 15, in which he will get out of the dream and win the game. Oh, and watch out for the slimes. There are slimes that phase through walls hunting the player. On death, the player will spawn at the beginning of the maze.

**Installation/Start Up**

To run the game, open the Microsoft Visual Studio solution and use the local Windows Debugger to compile and run the program. The program successfully runs every time with the version on the machines in the Game Dev Lab.

**Rules: How to play the game**

The game doesn’t have very many rules. The entire premise of the game is to navigate through each maze and find the end pillar leading to the next maze. The only Rules of the game are that you cannot move outside the walls of the maze. Other than that you can choose to do whatever you want, but be warned…. You are being timed.

**User Options:**

The user has some very simply but in-depth decisions. Firstly, the user decides where to move. Obviously the user must somehow navigate to the end of the maze. The second decision is how to use stamina. Sprinting can decrease your time greatly, however jumping can give hints on if you are on the right path or not. If you are face-to-face with a slime and out of stamina you could die, which would send you back to the beginning of the maze, so you never want to be too low on stamina.

**Tutorial Usage**

We used a lot of material from the tutorials. Rendering the skybox, texture mapping for textures and text on screen was all done using the same methods as learned in the tutorials. The walls are rendered using the methods learned in 6. We also used the *Arraylist* class to hold our nodes and edges; however we did make slight changed as to what we required.

**Interactions:**

There are only a few Types of interactions a player can have I the game. If a player runs into a wall they collide with it. Collision with the walls is calculated in 2D so even if you manage to jump high enough to go over the walls you cannot physically go over them.

The other type of interaction is with the *Slime* enemy. The slime enemies move in a random path and don’t collide with walls. So you need to be careful one doesn’t sneak up from behind. If the player finds itself trapped by an enemy, the player can jump on it to destroy it. If a player succeeds in jumping on the slime, he gets a short boost in the air. If the player gets hit by the slime when not jumping, he “dies” and is sent back to the start of the level.

**Game objectives:**

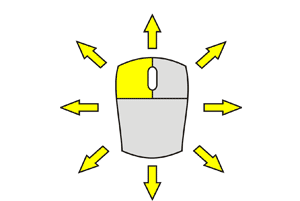
The objective of the game is to complete all 15 mazes in the shortest amount of time.

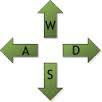
The player has to navigate through the mazes avoiding or destroying slimes to find the end gate and exit all 15 mazes.

**Controls:**

The controls of the game are quite simple.

Move: W A S D Look: Mouse Jump: Space Bar





If a player would like to place a save point they can do so by pressing the **F** key. At any point a player may return to the last placed save point by pressing the **G** key.

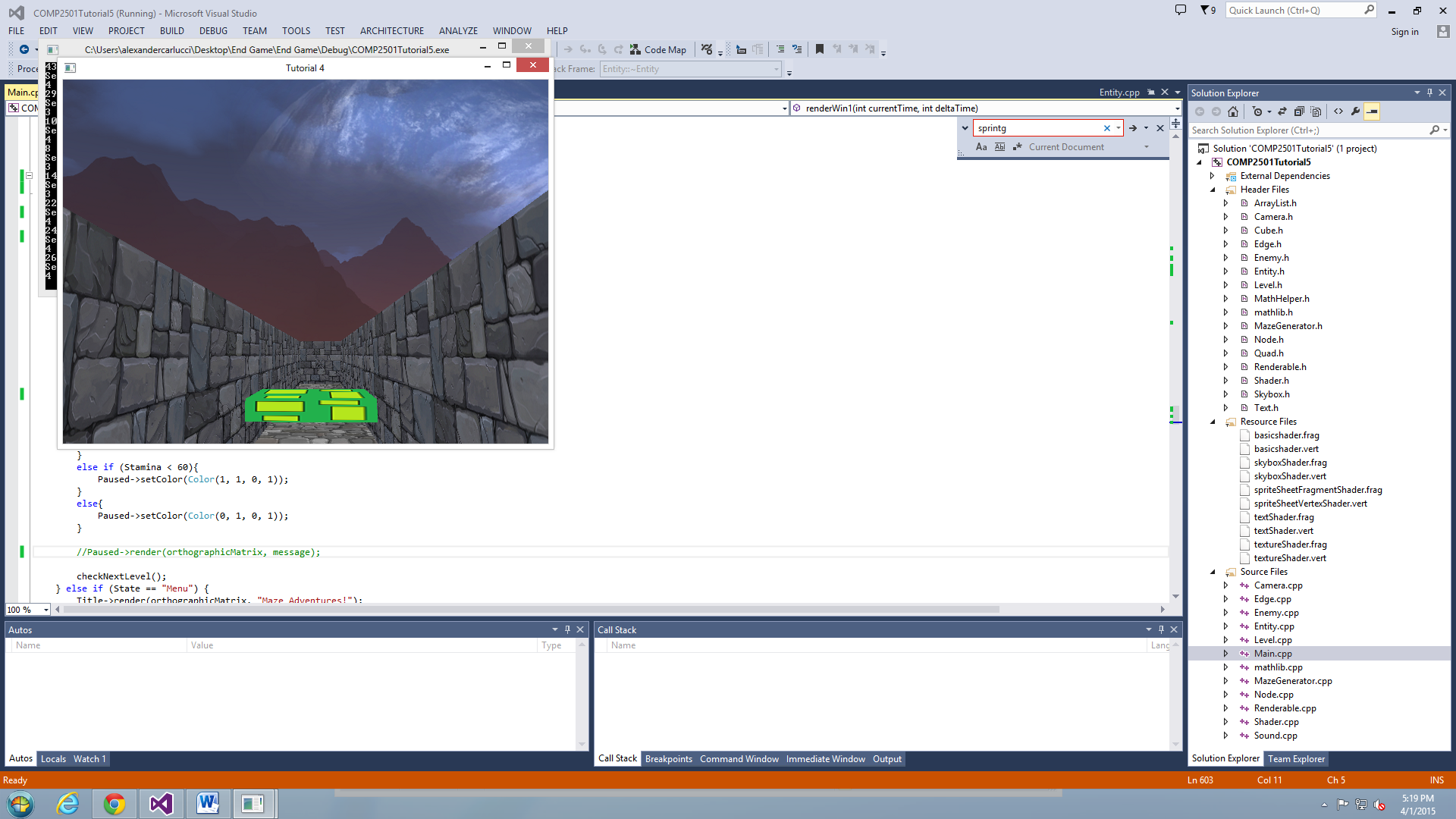
If you hold the **SHIFT** button you “Sprint”, giving you a small movement speed buff. While sprinting or jumping your stamina depletes. To jump you need a minimum of

five stamina.

These Controls cannot be changed but are viewable in the *controls* section of the menu screen.

**Levels**

There are a total of 15 levels in the game. As the player progresses through a level the next one is generated a bit bigger. As well as increasing in size the amount of enemies that spawn in each level increases with the level number. To complete a level the player must navigate through the maze and find the exit marked by a tall pillar. Each level is randomly generated. We used a simple depth first search algorithm explained better in the Programmer Guide section of the report.



**Play**

When playing the game make sure to keep an eye on your stamina. If you drop below 5 and encounter slime you will not be able to jump and this may lead to your death.

Use sprit sparingly. Jumping is a more useful way to spend your stamina. You can use your jump to look over walls for a short period of time and find out which direction you should go.

**Score**

There is no ‘score’ in the game; however the player can mark their accomplishments in two ways. Firstly, there is an overall time is has taken the user. The time simply constantly goes up as the user plays the game. If one player beats all 15 levels with a lower time than the next, they can say that they beat the game quicker, and therefore better, then someone with a longer time. The second is the amount of levels the player has accomplished. The more levels you beat the better you are doing. The last level, level 15, is a 21 by 21 maze. This is very difficult to solve and requires patience and strategy to execute properly.

**Programmer Guide**

**Software Architecture**

The software architecture of the game is not too complicated; however it can be hard to follow as we did not take full advantage of object-oriented programming. The control flow is entirely done in the main class. The *renderwin1* function renders certain things based on the current game state. There are 5 game states: play, menu, controls, loading and game\_finished. The main class has an array of levels. Each level has an Arraylist of randomly generated nodes and edges. In the ‘play’ state, a simple call to each level in the levels Arraylist will tell each level to draw itself. Main also has an Array of 30 enemies, but only a certain amount are rendered based on the current level the player is on. The player is the just the camera as the game is first person. There is a character model, as when you look down you can see red legs running when you run. There are three different collision types in the game. The first kind is player-wall collision. This is handled by pushing the player towards to closest edge in the graph the exact distance he is away from the nearest edge. The second is the player-finish collision. This is just distance from a point to a point. If the player is within a certain distance of the end, he has won. The same type of collision is done for our last type of collision, which is player-enemy collision. All the collision is done in main. The Enemy class is a sub class of Entities, so each enemy can draw itself. The player just progresses through each level, between each the ‘loading’ state is displayed. The user can press p to go to the menu state, where he can resume play or go to the credits or controls.

**Data Structures and Performance**

The main type of data structure we used was the Arraylist that was provided for us. This was used by the Level class to hold the nodes and edges of each level. With respect to performance, there were not many issues except that we quickly realized two things were very costly. Firstly, the square root functions running many times each time the renwin1 function is called is very costly. To address this we just removed the square root, because it is just not needed. Secondly, for error checking purposes we would print things to the console output. This is also very costly, so removing the output one we were done error checking a certain part of the code kept things running smoothly.

**External Libraries**

The only external library we used was FMOD, which we uses for the sound in our game.

**Tools**

We hardly used any tools to help create our game. Everything was out of primitives so we didn’t have to import anything from a 3D modeling program or anything. The artwork was taken off of the internet so we cannot make money of the game as it is now, unless we make our own art (walls, floors, and skybox). The only tool we used was MS Paint, to convert images to PNG and resize when necessary.

**Difficult Parts**

There were not too many difficult parts of the game. The three main parts were randomly generating the nodes and edges, rendering the walls correctly, and collision detection. We accomplished all of these tasks. One thing that we did get stuck on however is importing external dependencies correctly. Hours were spent getting silly linker errors that were frustrating.

**Our game is unique because:**

Randomly generated levels: no matter how many times you play the game it’s different every time. Random generation is not easy and drawing a level that has been randomly generated is a tall task. But we feel like we hit the nail on the head and this is what puts our game apart from the rest.

Ominous/ Almost creepy feel: We decided to go for not scary but creepy sound and art style. You’re in a dream so it doesn’t need to look realistic. Everything is in darker color and the music makes you feel like you’re not in a safe place.

**Our hopes for the game in the future:**

There were a couple of items that we wanted to touch up before handing the game in. some of these include:

Saving high scores- we really wanted to have people compete against each other to see how long it takes to complete our game.

Texturing the player: we wanted to get a texture to put on the player model, but since we designed our own we couldn’t find any free resources online and had to create one. So we ended up using a basic shader to color him red for the time being.

AI: we wanted to have the slimes have a bit more intelligent AI but since AI takes a lot of time to make we decided to polish the rest of our game up before attempting this.

Camera: we wanted to have a Third person camera following the player, but since we have tight corridors the math behind a smart camera would have taken too long to complete and likely resulted in our game being incomplete.– we ended up going with first person