Matthew Yanzer

Senior Project Defense Document for "Late Night Snack"

Matthew Yanzer

CSCI 499-Senior Project Defense

Dr. Paul West

Due April 8, 2021

Table of Contents

Chapter 1: Statement of Purpose

Ever since coming to Charleston Southern University, I have been fascinated by game design and have always wanted to develop a game myself. However, I began my college career with no former experience in game design or computer science in general. Since game design was one of the major fields I wanted to pursue as a vocation, I knew that I would need a way to take what I learned and apply it to developing games. Thus, my senior project involves a simple game that runs in a C++ engine and utilizes vector graphics to display to a command terminal. Since a lot of the courses taught here utilize C++ for assignments and projects, I figured that building a simple game in C++ would be a good way to begin gaining experience in game design, learn new methods of coding for general computer science application, and apply formerly learned methods from the classes I have taken here.

Chapter 2: Research & Background

A large portion of the research that went into the development of this project was finding a proper engine to build, compile, and run the game. My initial plan was to utilize Unreal Engine, a free online game engine that utilizes both C++ and a unique coding template called Blueprints, to construct the mechanics of the game and Blender, a 3D modeling software, to build the models for the player, threat, and environment. However, that plan was scrapped as Unreal Engine, despite being a good tool to construct 3D games, required a large set of libraries and functions to allow coding in C++ to work. Given the time constraints of the project and the extensive time needed to properly learn how to utilize Unreal Engine's libraries, I decided to move my project to a simpler system that would allow me to focus more on efficient mechanics construction and more intuitive C++ coding.

While searching for a simpler system, I came across a Youtuber by the name of OneLoneCoder who makes videos about developing games within C++ and running them in console. His engine, referred to as the olc Console Game Engine, presented an environment that suited my project's needs. In addition, one of his tutorials for developing games in his engine was a first-person shooter game that made use of vector graphics to display to the console. Using his tutorial as a base, I figured that with some modifications to the code, I could easily see my project through.

Chapter 3: Project Language(s), Software, and Hardware

This project utilizes only C++ for its construction. No additional languages are needed as the entire game runs in a single console. I chose C++ for two reasons. First, C++ is my favorite language. I have more experience in C++ than in any other language, making it my first choice for both personal and academic projects alike. Second, the olc Console Game Engine runs exclusively in C++, making the use of any other language for this project pointless and inefficient.

For software, I used Visual Studio 2017 to compile and run my project. Visual Studio is a free software used for editing, compiling, and running code, making it very useful for testing my project and providing an easy-to-use interface for constructing my project. All of the project's files were compiled in Visual Studio. In addition, the project uses OneLoneCoder's olc Console Game Engine, which is a C++ file kept within the same folder as the game. It translates certain functions specific to the engine, which include functions for displaying text and graphics to the screen. Another program this project uses is OneLoneCoder's sprite editing program, which was also coded in C++. It allows the user to make basic .spr files, which can then be referenced by the olc Console Game Engine to add more colorful graphics to console games using that engine.

Special hardware besides a standard PC is not required for this project. However, it is important to note that the PC needs to run on Windows 10 in order for the program to be runnable.

Link to OneLoneCoder's tutorial: https://www.youtube.com/watch?v=xW8skO7MFYw

Link to Visual Studio's website: https://visualstudio.microsoft.com/

Link to olc Console Game Engine:

https://github.com/OneLoneCoder/videos/blob/master/olcConsoleGameEngine.h

Link to OneLoneCoder's first-person shooter:

https://github.com/OneLoneCoder/videos/blob/master/OneLoneCoder_ComandLineFPS_2.cpp

Link to OneLoneCoder's sprite editor:

https://github.com/OneLoneCoder/videos/blob/master/OneLoneCoder_SpriteEditor.cpp

Chapter 4: Project Requirements

Notes:

- Requirement Importance Range: 1-5. 1-2 is completely optional and likely are only visual improvements. 3-4 is a necessary component that can be modified or minimized as long as primary functionality remains intact. 5 is essential and will likely not be changed during the development cycle.
- Threat is a term that refers to the AI that will chase the player and cause a "Game Over" instance.
- Character is the manifestation of the role the player assumes in the game.
- Functional Requirement Type List
    - Functional: Requirement that is an entity or event that progresses the game in some manner.
    - Performance: Requirement that should be optimized to account for performance drops.
    - Visual: Requirement that allows the system to be more user friendly and pleasing to the eye.
- Secret objective refers to the ending where the player slays the threat and wins the game.
- Main objective refers to the ending where the player buys the soda and retreats to the exit to win the game.

Requirements:

1. Main Menu
2. Control Menu
3. Pause Menu
4. Character Class
5. Hiding Box Class
6. Green Vending Machine Class
7. Blue Vending Machine Class
8. Key Class
9. Sword Class
10. Alarm Class
11. Campus Environment
12. Threat Class
13. Game Over Menu
14. Victory Menu
15. Secret Victory Menu
16. Exit Door Class

1) Requirement Name: Main Menu

   Requirement Importance: 4

   Requirement Type: Visual, Functional

   Description: A simple interface that allows players to start their game or refer to the game's controls.

   Rationale: This interface starts the game.

   Fit Criterion: Allows player to start game.

   Dependencies: Controls Menu, Single Player Functionality

   Conflicts: N/A

2) Requirement Name: Controls Menu

   Requirement Importance: 3

   Requirement Type: Functional, Visual

   Description: A controls menu for the player to learn how to play the game.

   Rationale: This menu displays the controls of the game.

   Fit Criterion: Control over Resolution

   Dependencies: Main Menu

   Conflicts: N/A

3) Requirement Name: Pause Menu

   Requirement Importance: 3

   Requirement Type: Functional, Visual

   Description: A menu that pauses the game and allows the player to end the game early.

   Rationale: This menu allows the player to pause the game.

   Fit Criterion: Pauses the game, able to quit to main menu

   Dependencies: Main Menu

   Conflicts: N/A

4) Requirement Name: Character Class

   Requirement Importance: 5

   Requirement Type: Functional, Performance

   Description: A character class that allows the player to interact with the game.

   Rationale: This class is the role the player assumes in the game.

   Fit Criterion: This class will have attributes for a character.

   Dependencies: Flashlight Class, Key Class, Sword Class

   Conflicts: N/A

5) Requirement Name: Flashlight Class

   Requirement Importance: 4

   Requirement Type: Functional, Visual

   Description: A flashlight class that will allow the player to toggle the rendering distance of the game for better visibility, simulating a flashlight's effect in a dark environment.

   Rationale: This class allows the player to have a clearer range of visibility.

   Fit Criterion: An object that provides better visibility.

   Dependencies: N/A

   Conflicts: N/A

6) Requirement Name: Hiding Box Class

   Requirement Importance: 4

   Requirement Type: Functional

   Description: An area where the character can hide from the threat without getting caught.

   Rationale: This class allows the player to avoid the threat easily.

   Fit Criterion: A tile of the map that does not allow the threat to find the player.

   Dependencies: N/A

   Conflicts: N/A

7) Requirement Name: Green Vending Machine Class

   Requirement Importance: 5

   Requirement Type: Functional, Visual

   Description: A vending machine that allows the player to purchase a soda for $1.00.

   Rationale: Provides the player with a location to progress towards completing the objective of the game.

   Fit Criterion: An object that allows the player to search for change.

   Dependencies: N/A

   Conflicts: N/A

8) Requirement Name: Blue Vending Machine Class

   Requirement Importance: 5

   Requirement Type: Functional, Visual

   Description: A vending machine that allows the player to collect $0.25.

   Rationale: Provides the player with a clear location to progress towards completing the objective of the game.

   Fit Criterion: An object that allows the player to buy a soda.

   Dependencies: Couch Class

   Conflicts: N/A

9) Requirement Name: Key Class

   Requirement Importance: 4

   Requirement Type: Functional, Visual

   Description: A small key that the character can obtain to pursue the secret objective.

   Rationale: Provides the player with an object necessary to complete the secret objective.

   Fit Criterion: An object that allows the player to pursue the secret objective.

   Dependencies: Secret Door Class

   Conflicts: N/A

10) Requirement Name: Sword Class

   Requirement Importance: 4

   Requirement Type: Functional, Visual

   Description: A sword the character can use to slay the threat when the threat is close to the character.

   Rationale: This object provides the player a means to complete the secret objective.

   Fit Criterion: An object that can defeat the threat and complete the secret objective.

   Dependencies: Threat Class, Secret Victory Menu

   Conflicts: N/A

11) Requirement Name: Alarm Class

Requirement Importance: 3

Description: A pole with a button that the character can activate to ward off the threat.

Rationale: This alarm allows the player to defend themselves against the threat.

Fit Criterion: An object that sends the threat to the opposite corner of the maze and can be used only once.

Dependencies: Threat Class

Conflicts: N/A

12) Requirement Name: Campus Environment

Requirement Importance: 5

Requirement Type: Functional, Visual

Description: The area the character navigates to complete the objectives of the game.

Rationale: This is needed for the player to have an area wherein he can complete the objectives of the game.

Fit Criterion: An object wherein the player can navigate.

Dependencies: N/A

Conflicts: N/A

13) Requirement Name: Threat Class

    Requirement Importance: 5

    Requirement Type: Functional, Performance, Visual

    Description: A class that interacts with the player and causes the player to fail their current objective.

    Rationale: This class exists to pose a challenge to the player while they attempt to complete their objectives.

    Fit Criterion: An object/AI that causes the game to end if it interacts with the player.

    Dependencies: N/A

    Conflicts: N/A

14) Requirement Name: Door Class

    Requirement Importance: 4

    Requirement Type: Functional, Visual

    Description: A Class that allows the player to enter/exit different areas.

    Fit Criterion: This class is necessary for the player to access the different areas of the game.

    Dependencies: N/A

    Conflicts: N/A

15) Requirement Name: Game Over Menu

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: A menu that appears if the character fails his objective and that allows the player to start a new game or exit to the main menu.

Rationale: This menu is necessary so that the player can continue playing if he fails the objective or wishes to leave the game.

Fit Criterion: An interface that links to the main menu and that can start a new game.

Dependencies: N/A

Conflicts: N/A

16) Requirement Name: Victory Menu

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: A menu that appears if the character completes the main objective.

Rationale: This interface allows the player to start a new game or exit to main menu. It also serves the purpose of congratulating the player for completing the main objective.

Fit Criterion: An interface that links to the main menu, that can start a new game, and that congratulates the player.

Dependencies: N/A

Conflicts: N/A

17) Requirement Name: Secret Victory Menu

Requirement Importance: 3

Requirement Type: Functional, Visual

Description: A menu that appears if the character completes the secret objective.

Rationale: This interface allows the player to start a new game or exit to main menu. It also serves the purpose of congratulating the player for completing the secret objective.

Fit Criterion: An interface that links to the main menu, that can start a new game, and that congratulates the player.

Dependencies: N/A

Conflicts: N/A

18) Requirement Name: Exit Door Class

Requirement Importance: 5

Description: The player must interact with this door after acquiring a soda from a green vending machine to complete the main objective.

Rationale: This class allows the player to complete the main objective once he collects a soda from a green vending machine.

Fit Criterion: An object that the player can interact with to complete the main objective after acquiring a soda from a green vending machine.

Dependencies: Victory Menu

Conflicts: N/A

19) Secret Door Class

Requirement Importance: 5

Description: The player must interact with this door after acquiring the key to complete the secret objective.

Rationale: This class allows the player to access the sword class once he collects the key.

Fit Criterion: An object that the player can interact with to obtain the sword class after finding the key.

Dependencies: N/A

Conflicts: N/A

Chapter 5: Project Implementation Description & Explanation

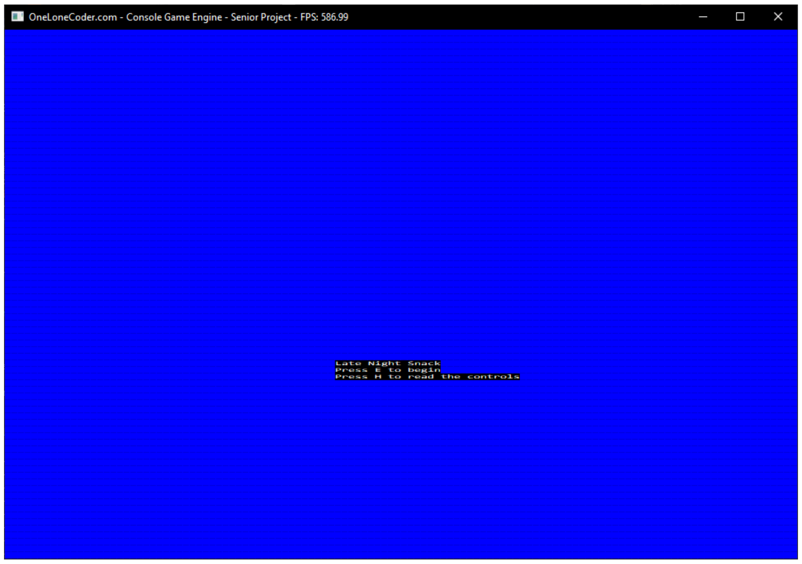
The project, put simply, is a game programmed in C++ that utilizes vector graphics and displays in the console of a Windows machine. The objective of the game is simple. The player takes the role of a poor college student who needs to collect enough change to buy a soda from a vending machine. The student, since he has no money, must scrounge throughout the campus (represented by the maze) to search blue vending machines for change. The player is equipped with a flashlight, which changes the rendering distance of the game and allows the player to see further. Once the student has collected $1.00 total, he can head to the green vending machine on the left side of the maze and buy a soda. From there, the student can head toward the exit at the bottom of the map and leave the maze.

However, there is a monster in the maze searching for the player, and the player needs to move strategically in order to avoid being captured. If the player is caught by the monster, the player receives an instant game over. There are two ways the player can evade the monster while trying to escape. Throughout the maze are littered hiding spots the player can run into. While in a hiding spot, the player cannot be caught by the monster. In addition, there are four emergency poles at the corners of the maze. If the player activates a pole while the monster is nearby, the monster is teleported to the opposite side of the map, losing track of the player if it was chasing him. Each emergency pole can be used only once per game.

Although buying a soda and escaping the maze is the normal ending to the game, there is a second secret ending the player can also pursue. Within the maze a key is randomly moving about. If the player collects the key, he can open a cyan wall in the maze that conceals a sword. After obtaining the sword, the player can then chase down and slay the monster, thus obtaining the secret ending.

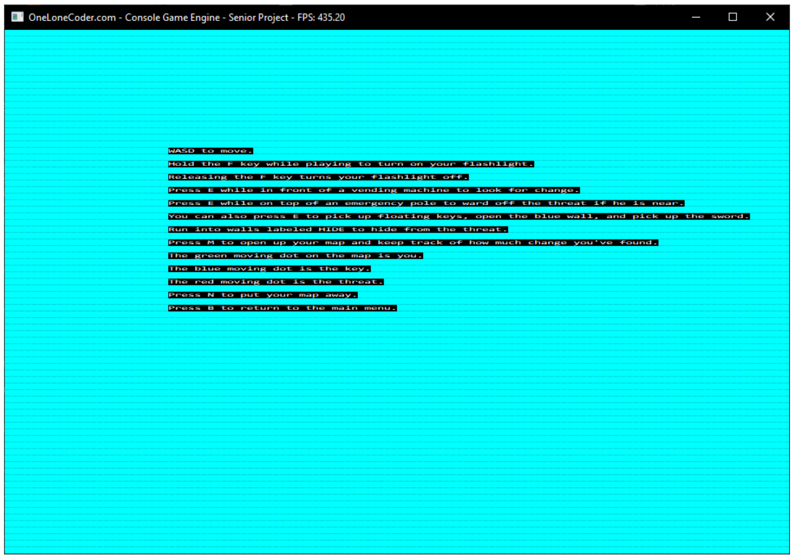
When compiling the game for the first time, the player is greeted to a main menu screen (See Fig. 1) that allows the player to start the game or to visit a page detailing how to play the game (See Fig. 2). Pressing the "E" key on the main menu will start the game, while pressing the "H" key will open the controls menu.

(Fig. 1: *The Main Menu*)



```
OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 586.99

Late Night Snack
Press E to begin
Press H to read the controls
```

(Fig. 2: *The Controls Screen*)\

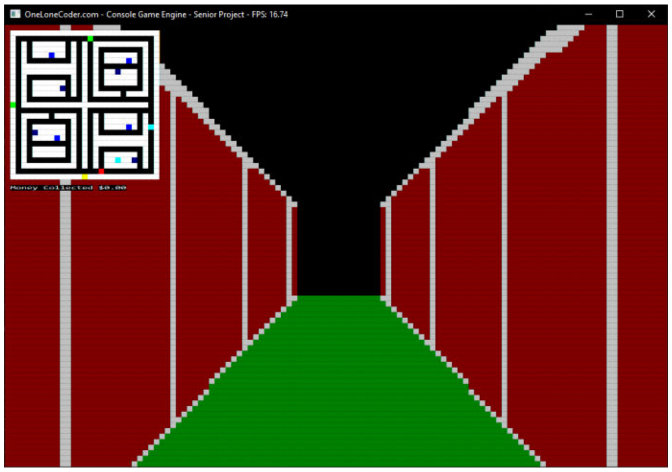


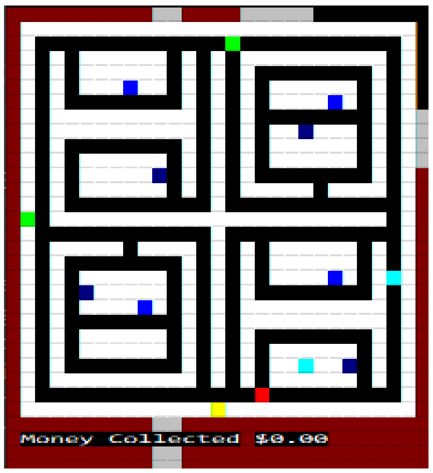
```
OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 435.20

WASD to move.
Hold the F key while playing to turn on your flashlight.
Releasing the F key turns your flashlight off.
Press E while in front of a vending machine to look for change.
Press E while on top of an emergency pole to ward off the threat if he is near.
You can also press E to pick up floating keys, open the blue wall, and pick up the sword.
Run into walls labeled HIDE to hide from the threat.
Press M to open up your map and keep track of how much change you've found.
The green moving dot on the map is you.
The blue moving dot is the key.
The red moving dot is the threat.
Press N to put your map away.
Press B to return to the main menu.
```

Once the game is started, the player will be presented to a first-person view of a maze with a small map of the entire maze in the top left corner of the screen as well as a display stating the total amount of money the player has currently (See Fig. 3a and Fig. 3b). The map displays the paths of the maze with black squares and marks the current location of the player with a green square on the paths. In addition, a cyan square on the path marks where the key currently is, and a red square marks the current location of the threat. Alongside the map's paths are blue squares that represent vending machines, dark blue squares that indicate the location of hiding places for the player, a green square to the left of the map indicating where the green vending machine is, a yellow square marking the exit, and a cyan square in the bottom right quadrant of the map revealing the general location of the cyan wall.

(Fig. 3a (left): *Starting a new Game*)

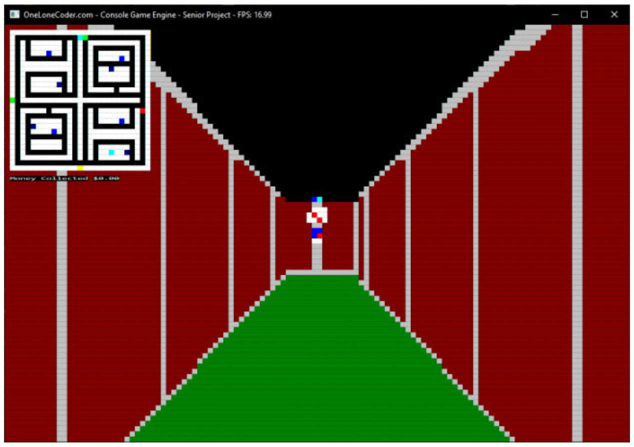
(Fig. 3b (right): *Map and Money Total*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 16.74
Money Collected $0.00



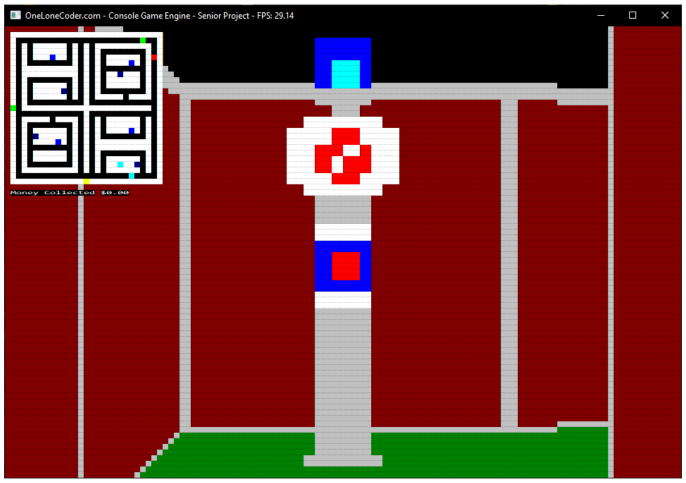
The player can hold the "F" key to toggle his flashlight, enabling him to see further down long corridors in the maze. Mechanically, there is a variable that represents the rendering distance of the game related to the player's position, determining how far the player can see. By holding down the "F" key, the player's rendering distance is increased, thus enabling him to see further. Figure 4 demonstrates this by showing the same hallway as Figure 3 with the flashlight active.

(Figure 4: *The Starting Corridor with the Flashlight on*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 16.99
Money Collected $0.00

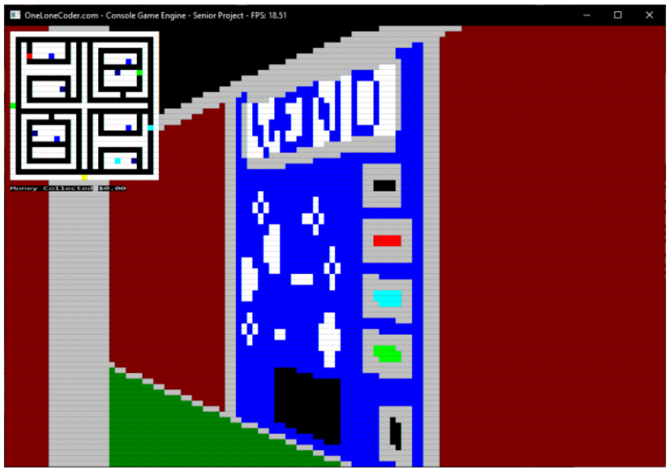
As the player navigates the maze, he will encounter emergency poles at the corners of the map that can be used to deter the threat (See Fig. 5). By pressing the "E" key while standing on one when the threat is in nearby, the threat will instantly teleport to the opposite side of the map. Each emergency pole can only be used once per game, making it important for the player to reserve using them for only dire circumstances.

(Fig. 5: *Emergency Pole*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 29.14
Money Collected $0.00

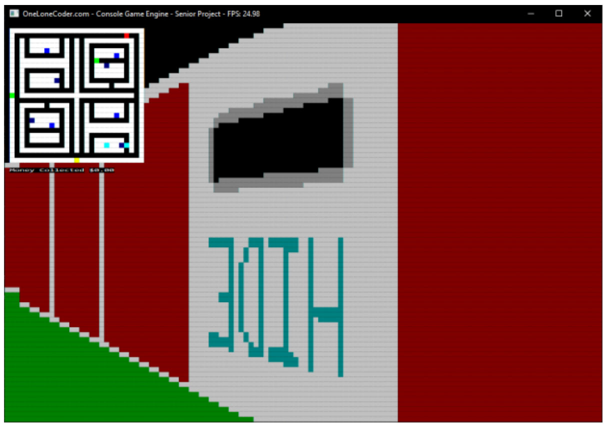
In addition to emergency poles, the player will find blue vending machines (See Fig. 6) that contain change. By pressing the "E" key in front of one, the player can collect $0.25 from it once per game. There are four blue vending machines around the map, and the player will need to visit them all to collect enough money to buy a soda.

(Fig. 6: *Blue Vending Machine*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 18.51
Money Collected: $0.00

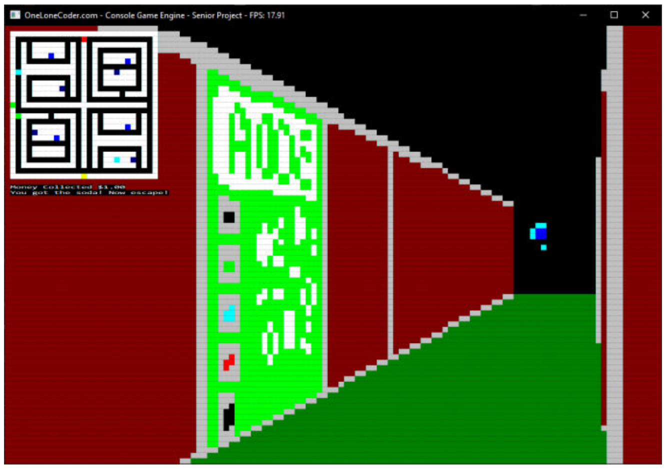
If the player is being chased by the threat and an emergency pole is not nearby, the player can run into a hiding place and wait for the threat to pass by (See Fig. 7). Four hiding places are scattered throughout the map, with only one in each quadrant of the map.

(Fig. 7: *Hiding Place*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 24.98
Money Collected: $0.00

Once the player has collected $1.00, he can collect a soda from the green vending machine (See Fig. 8). The green vending machine cannot be searched for change. Once a soda is purchased, a message beneath the map will encourage the player to find the exit now that the soda has been purchased. Before purchasing a soda, the player cannot leave the maze.

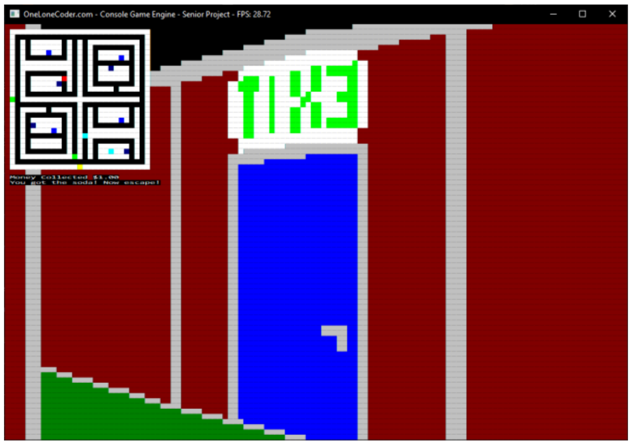
(Fig. 8: *Green Vending Machine*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 17.91

```
Money Collected $1.00
You got the soda! Now escape!
```

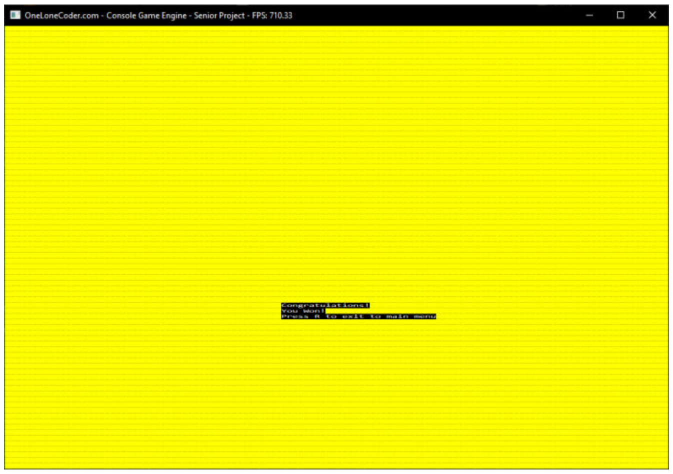
If the soda has been purchased, the player now needs to find the exit (See Fig. 9). After pressing the "E" key in front of the exit, the player will exit the maze and be presented with the normal victory screen (See Fig. 10). The route the player took in this scenario is the way to obtain the normal ending to the game and the normal victory screen.

(Fig. 9: *Exit Door*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 28.72

```
Money Collected $1.00
You got the soda! Now escape!
```

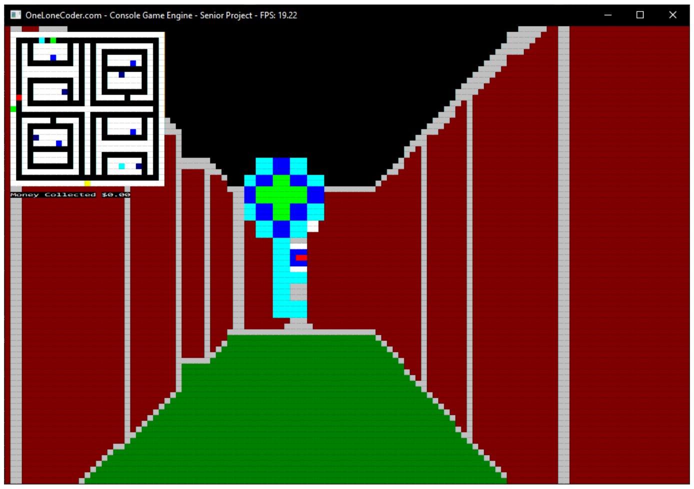
(Fig. 10: *Normal Victory Screen*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 710.33

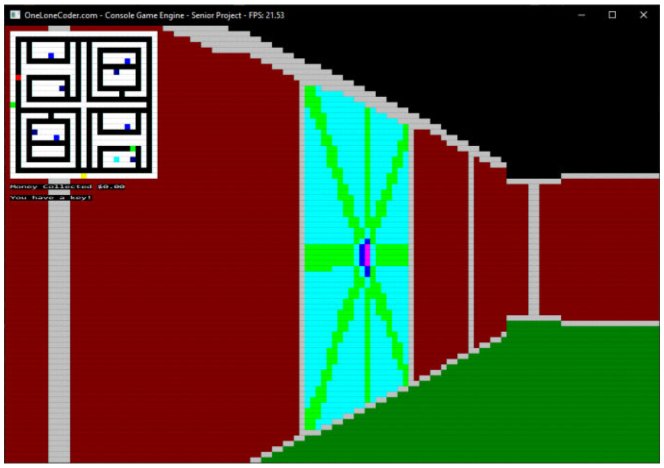
```
Congratulations!
You Won!
Press R to exit to main menu
```

As mentioned previously, there is a second way to beat the game that involves finding a sword to slay the threat. To beat the game this way, the player must first find a cyan key floating through the maze (See Fig. 11). Once this key is found, the player then needs to maneuver to a cyan wall somewhere within the maze. The wall can be found near the cyan square on the map that does not move (See Fig. 12).

(Fig. 11: *Cyan Key*)

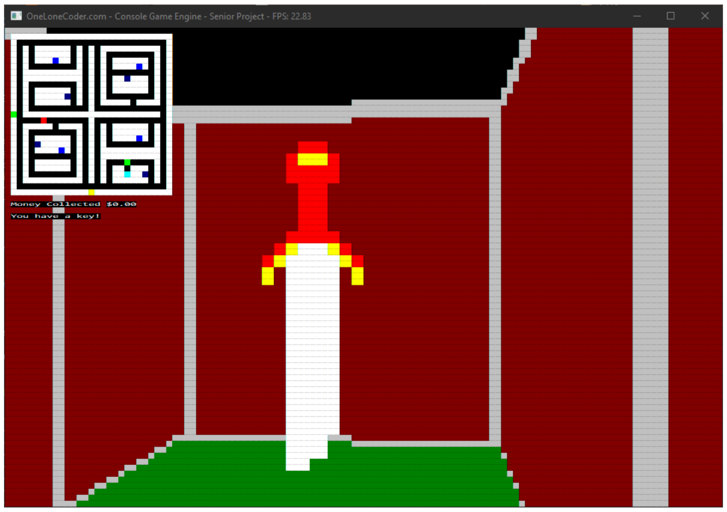

OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 19.22
Money Collected $0.00

(Fig. 12: *Cyan Wall*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 21.53
Money Collected $0.00
You have a key!

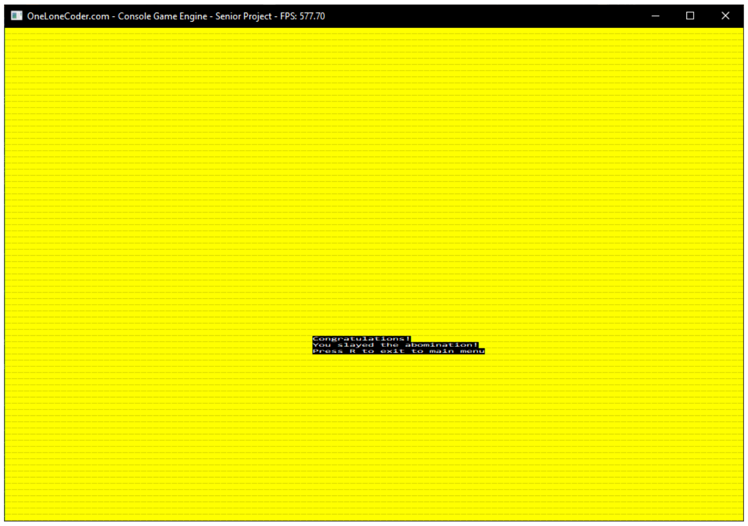
If the player has the cyan key, he can press the "E" key in front of the cyan wall to remove the cyan wall, revealing a sword behind it (See Fig. 13). The player can pick up this sword by standing over it and pressing the "E" key.

(Fig. 13: *Sword*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 22.83
Money Collected $0.00
You have a key!

After obtaining the sword, the player can then chase down the threat. If the player runs into the threat with the sword, the player slays the threat and obtains the secret victory screen (See Fig. 14).

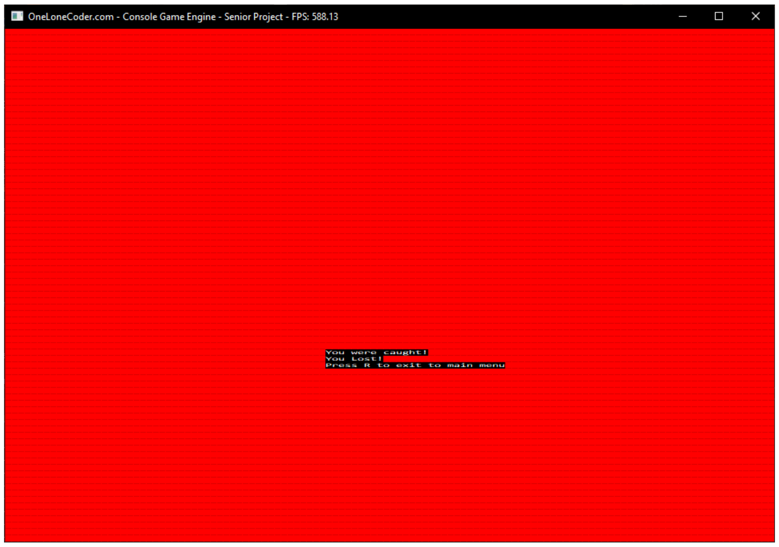
(Fig. 14: *Secret Victory Screen*)


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 577.70

```
Congratulations!
You slayed the abomination!
Press R to exit to main menu
```

If at any time during the game the player runs into the threat without the sword, the threat kills the player and instantly triggers a game over screen (See Fig. 15). If this happens, the player will have to start a new game from the beginning if he wants to try to win again.

(Fig. 15: *Game Over Screen*)

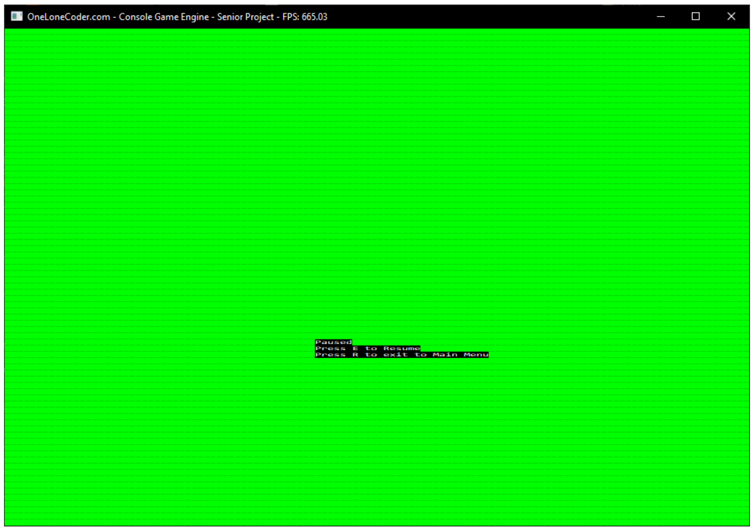

OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 588.13

```
You were caught!
You Lost!
Press R to exit to main menu
```

If at any time during gameplay the player wants to take a break, the player can pause the game by pressing the "P" key. This will present the player with the pause screen (See Fig. 16).

Pressing the "E" key while on the pause screen will resume gameplay, while pressing the "R" key will send the player back to the main menu screen.

(Fig. 16: *Pause Screen*)



```
Paused
Press E to Resume
Press R to exit to Main Menu
```

OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 665.03

It is important to remember that the game will keep track of the player's keyboard input so long as the program is running, regardless of whether the window is minimized.

Link to the source code repository: https://github.com/MattYanzer/SeniorProjectMatthewYanzer

(Bear in mind that the GitHub repository is private. Send me an email containing your GitHub username if you wish to access it and I will make you a collaborator.)

Chapter 6: Test Plan

Testing for this project was carried out in two ways: unit/integration testing and playtesting. During integration testing, I programmed the game to begin in certain states to simulate scenarios that a player would run into during a game to ensure that the game behaved as I expected it to in those scenarios. A lot of these tests revolved specifically around the AI for the threat to ensure that the threat behaved properly when navigating the maze. Examples of such tests include ensuring that the AI can turn and proceed down a different corridor when presented with an intersection, ensuring that the AI will randomly pick one of two corridors when presented with a three way intersection and not proceed down the corridor it came from, ensuring that the AI would pursue the player if the player was seen, and ensuring that the player could hide from the threat without the threat being able to enter the hiding place. In addition, an integration test was applied to the player, namely whether the AI could be slain once the player had the sword. All of these tests were assigned specific time limits to ensure that each result was happening promptly and as expected.

For playtesting, I approached five of my peers and asked them to run through my program multiple times, answering a series of yes or no questions during their experience to ensure that the program was behaving as it should. Their answers were then documented and placed within a chart to determine if each of them was sharing the same expected experience with the game or if any of them experienced bugs within the game that prevented them from playing the game as intended or at all.

Here is a list of the integration tests I ran while testing the project:

AI Testing

1. When encountering a West/South intersection from the west, does the threat turn toward and move down the south path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

2. When encountering a West/South intersection from the south, does the threat turn toward and move down the west path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

3. When encountering an East/South intersection from the east, does the threat turn toward and move down the south path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

4. When encountering an East/South intersection from the south, does the threat turn toward and move down the east path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

5. When encountering a West/North intersection from the west, does the threat turn toward and move down the north path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

6. When encountering a West/North intersection from the north, does the threat turn toward and move down the west path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

7. When encountering an East/North intersection from the east, does the threat turn toward and move down the north path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

8. When encountering an East/North intersection from the north, does the threat turn toward and move down the east path? (Send AI toward intersection, wait for result after ten seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded.)

9. When encountering a three way intersection, does the AI pick a random direction (besides the direction it came from) and head in that direction? (Send AI toward intersection and wait for result after five seconds. If the AI has not turned in the expected direction, the test has failed. Otherwise, it succeeded. Run test multiple times to ensure that different directions are picked.)

10. If the AI sees the player, does the AI head toward the player's last seen position? (Give AI last seen coordinates with chase mode active and ensure that it catches player after ten seconds.)

11. If the threat sees the player and the player hides, is the threat prevented from catching the player? (Give AI last seen coordinates with chase mode active and ensure that it sees

player. Once player is seen, move Player into the hiding spot behind the player's previous location, and wait to see if the AI follows the player into the hiding spot. Give ten seconds for result. If the player is safe, the test succeeded.)

12. Does the AI travel down hallways in a straight line? (Wait ten seconds while letting the AI move in a hallway)

Player Tests

1. If the player has the sword, does the threat die when it runs into the player? (Have the AI run into the player while the player has the sword. Wait five seconds for result.)

Below is the sheet of questions that was used during playtesting:

[Mark each question as Y "yes" or N "no"]

a. Did the game open to the starting menu?
b. Did the game start upon hitting the start button?
c. Can you control the player?
d. Can you toggle the flashlight?
e. Does the threat kill you upon contact?
f. Can you hide from the threat?
g. Does your change total increase after interacting with a vending machine?
h. Are you presented with the Game Over screen upon player death?
i. Can you buy a soda when you have collected enough change?
j. Are you prevented from buying a soda if you do not have enough change?

k. Upon pressing E in front of the exit after buying a soda, are you presented with a win screen?

l. Can you pause the game?

m. Can you exit the game from the pause menu?

n. Can you resume the game from the pause menu?

o. Can you reach the Main Menu from the Game Over screen?

p. Can you pick up the key by pressing E while on its space?

q. While possessing the key, can you open the cyan door?

r. Once the door is open, can you pick up the sword?

s. If you have the sword, do you win after running into the threat?

Chapter 7: Test Results

In short, testing was a success across the board. All of the integration tests passed as expected, and all of the playtesting results succeeded unanimously. Below are a couple of charts detailing the results of both the integration tests and the playtesting results (The numbers of the tests correspond to the lists in Chapter 6):

	Attempt 1	Attempt 2	Attempt 3	Attempt 4
AI Test 1	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 2	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 3	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 4	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 5	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 6	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.

AI Test 7	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 8	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.	Yes; happened within required 10 seconds.
AI Test 9	Yes; happened within required 5 seconds. (Turned right.)	Yes; happened within required 5 seconds. (Went Straight)	Yes; happened within required 5 seconds. (Turned Right)	Yes; happened within required 5 seconds. (Went Straight)
AI Test 10	Yes; AI caught player within 10 seconds.	Yes; AI caught player within 10 seconds.	Yes; AI caught player within 10 seconds.	Yes; AI caught player within 10 seconds.
AI Test 11	Yes; Player was safe after 10 seconds.	Yes; Player was safe after 10 seconds.	Yes; Player was safe after 10 seconds.	Yes; Player was safe after 10 seconds.
AI Test 12	Yes; happened within 10 seconds.	Yes; happened within 10 seconds.	Yes; happened within 10 seconds.	Yes; happened within 10 seconds.
Player Test 1	Yes; AI was slain within 5 seconds.	Yes; AI was slain within 5 seconds.	Yes; AI was slain within 5 seconds.	Yes; AI was slain within 5 seconds.

Playtesting results (Y indicates a success; F indicates failure):

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
Subject 1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Subject 2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Subject 3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Subject 4	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Subject 5	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Chapter 8: Challenges Overcome

During the development of this project, there were several challenges I faced while attempting to get certain aspects of the code to work. One of the first challenges I needed to overcome was how to swap between gameplay and the various menus that needed to be included within the game. After giving it some thought, I reasoned that since the game is run with a while loop, I could activate and deactivate certain states of the game by using Boolean variables as flags to indicate which state the game was in. For example, a playing flag would be set to *true* during gameplay. Pressing the "P" key would switch the playing flag to *false* and the pausing flag to *true*. Since the gameplay logic was all contained within the playing flag, the variables would remain untouched until the playing flag was switched back to *true*.

Another difficulty I overcame was figuring out how to limit the amount of money a player could collect from a vending machine. Initially, the player's money variable could increase by .25 for every frame of the game, meaning that the player would get way more than $0.25 from a vending machine even if the "E" key was only held for a second. This problem was solved much like the previous one; I included Boolean variables that would keep track of which vending machines had been searched and would switch to *false* after being searched. That way, only one unit of .25 could be collected from each machine.

A third problem I encountered was figuring out how to program the AI for the threat to properly turn corners. Every time the AI would prepare to turn a corner, it would get caught on the inside wall of the turn and not move further. I eventually realized that the AI was turning as soon as the whole value of its coordinates matched the coordinates of the turn's tile, meaning that a large portion of the threat was not standing on the turn's tile yet and would run into the inside wall. I fixed this by having the AI's coordinates automatically adjust to the middle of the turn's tile so that the AI would not get caught on the inside wall anymore.

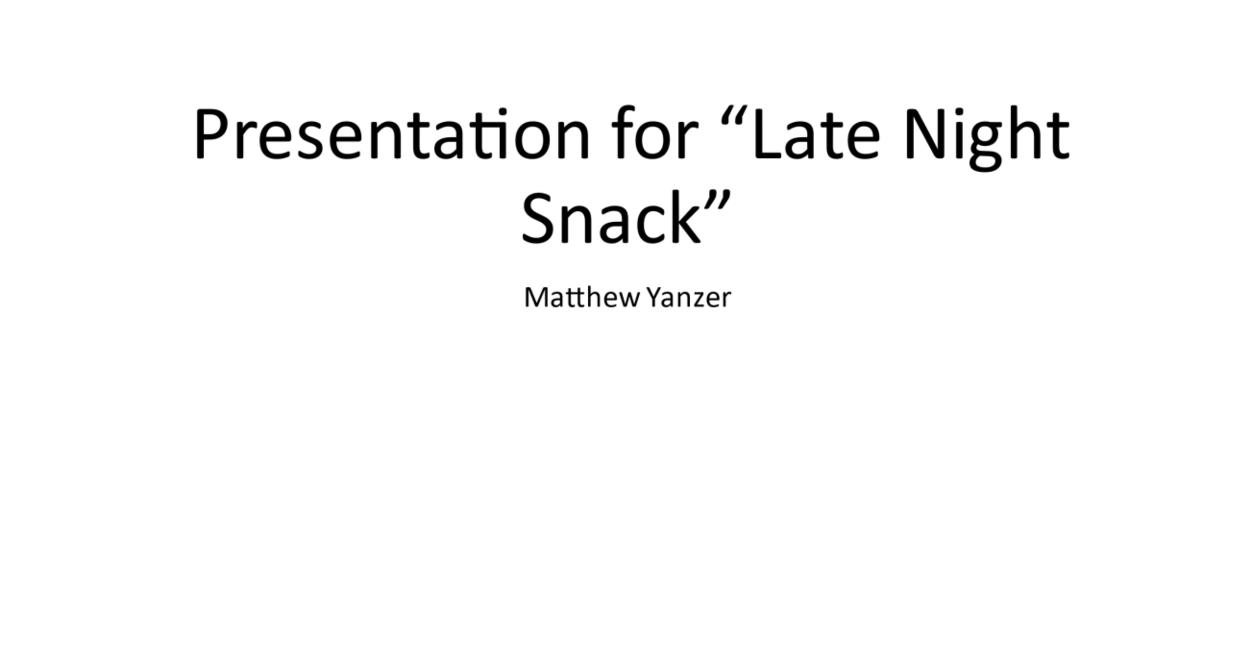
Lastly, I had a lot of difficulty preventing the AI's perception from extending beyond solid walls. A lot of times during early practice runs, the AI would see the player through a wall it was not supposed to see it through and start chasing the player. To solve this issue, I included a Boolean variable to check if the AI's perception had hit a wall. If the AI's perception hit a wall, the Boolean would be set to *false*, preventing the loop for the AI's perception from running for the rest of that frame of the game.

Chapter 9: Future Enhancements

Although the project is complete in its current state, there are a couple additional features I could add to make the game's experience more enjoyable. First, lowering the rate at which the player turns would make the player easier to control, especially for people who are new at the game. A fast turning speed, although efficient, can make it harder for people to accurately control the direction the player is facing, which could lead to the threat catching the player or similar frustrations. Such a task would be easy, as the rate at which the player turns is controlled by a simple trigonometric function. All that would need done is the altering of a couple of static values in the equation to make a slower, more reliable turning speed.

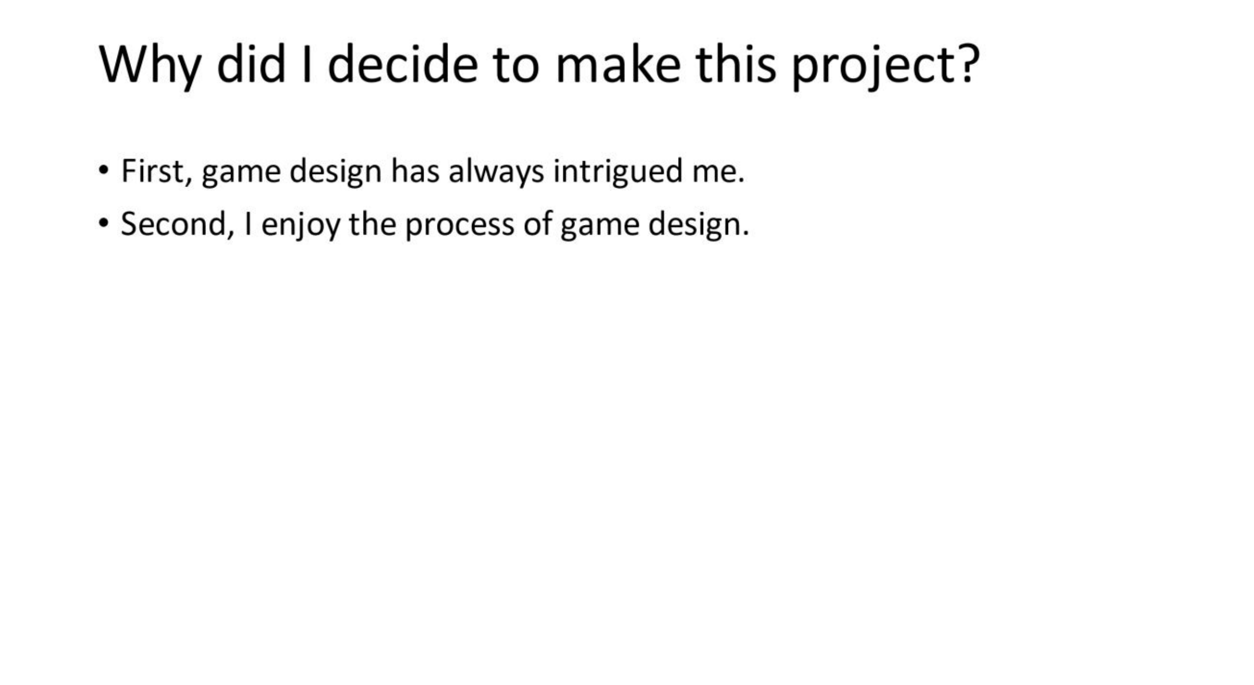
In addition, it is easy for the player to get caught on walls in the game's current state. This issue relates to how the game detects player collision against walls. When a player is holding the "W" key while intersecting a wall, the wall reverses the player's movement by an equal amount, effectively stopping the player. This mechanic can be frustrating for players who try to hug walls to minimize the time it takes to turn corners. A solution to this problem could be to alter the player's movement when hitting a wall at an angle rather than countering it to simulate the player moving along the wall.

Chapter 10: Defense Presentation Slides



Presentation for “Late Night Snack”

Matthew Yanzer



Why did I decide to make this project?

- First, game design has always intrigued me.
- Second, I enjoy the process of game design.

What is my senior project?(1)

- “Late Night Snack”
- Two goals
- Soda Ending
- Weapon Ending

What is my senior project?(2)

- Losing Conditions
- Random AI
- Chasing AI
- Alarm AI

Original Idea

- Blender
- Unreal Engine
- Blueprints

Why did my software change from Unreal to and Blender to C++?

- Lack of knowledge about Unreal.
- Difficult to test in.
- Time consuming.

How was C++ used?

- Threat AI
- Environment/maze
- Player's controls
- Interactive objects
- Menu selection

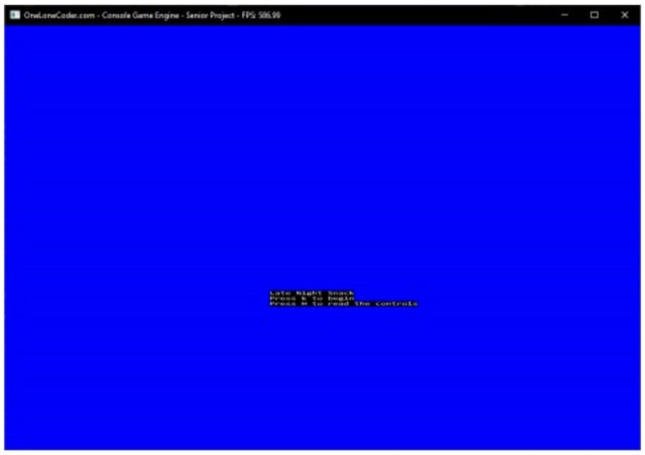
olc Console Game Engine

- Engine style
- Developed by One Lone Coder
- https://www.youtube.com/channel/UC -yuWVUplUJZvieEligKBkA

Sprite Editor

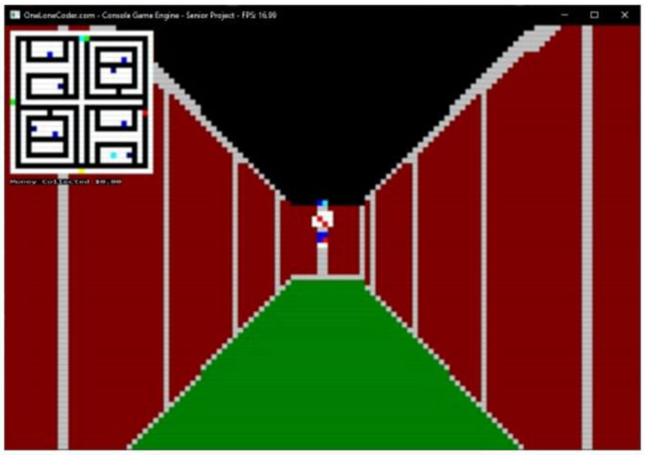
- Making sprites for the game
- Utilizes unique . spr file
- Also made by One Lone Coder

Screenshot of Title Screen

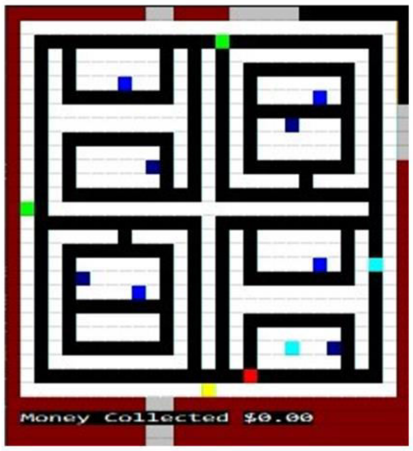

OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 586.99

```
Late Night Snack
Press S to begin
Press H to read the controls
```

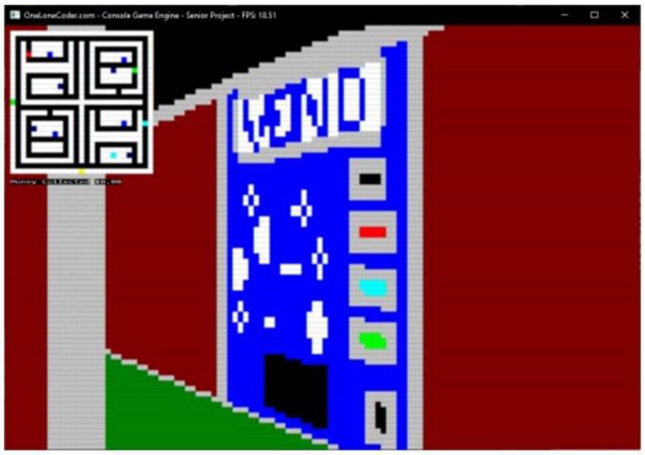
Screenshot of Gameplay


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 16.99
Money Collected $0.00

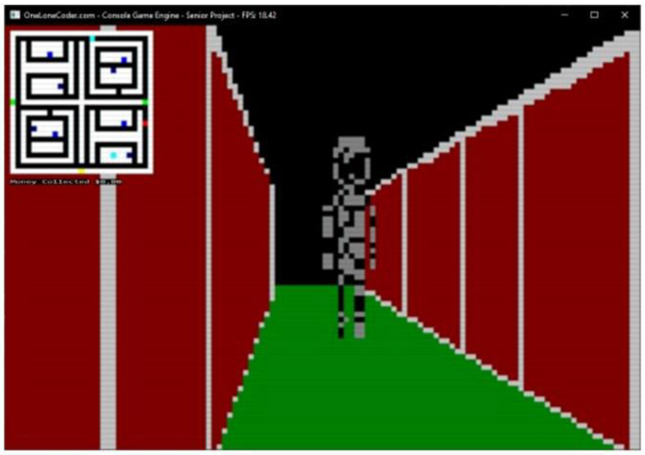
Screenshot of Map



Screenshot of Vending Machine


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 18.51
VEND
Money Collected $0.00

Screenshot of Threat


OneLoneCoder.com - Console Game Engine - Senior Project - FPS: 18.42
Money Collected $0.00

How did I test my project?

- My project was tested in two ways; unit testing and play testing.
- unit testing
- play testing
- Proper AI behavior
- Ability to properly control player
- Successful interactions with environment
- Correct displays