Snake Game

TeamPatrickByrnes Team 5

Daniel Donley, Sam Schultz, Jonah Nichols, Tyler Concannon

Project Report

Version: (1) **Date: (4/26/2019)**

Table of Contents

1 Change Log

This change log discusses any and all design changes from our Lab 10 and Lab 11 Submissions.

1.1 Lab 10 Submission

For our lab 10 submission, we got all of the funtionalities completed that were on the rubric. However, we could of done more on lab 10 to cut down on our work. For example, our snake would sometimes spawn out of bounds. In order to fix this, we had to fix the bounds on the random spawn of the snake so it would only appear in bounds. Another issue we had was when the snake would eat a pellet, another one would not respond. In order to fix this, we went into the randomizedPellet function. If the snake's location was on top of the pellet, it would then spawn another one. Finally, for the spawning of pellets, we did not have a timer in place. We simply made a timer set to 10 seconds, and if the snake does not reach the pellet by then, it calls the randomizedPellet function to respawn it in a different location.

1.2 Lab 11 Submission

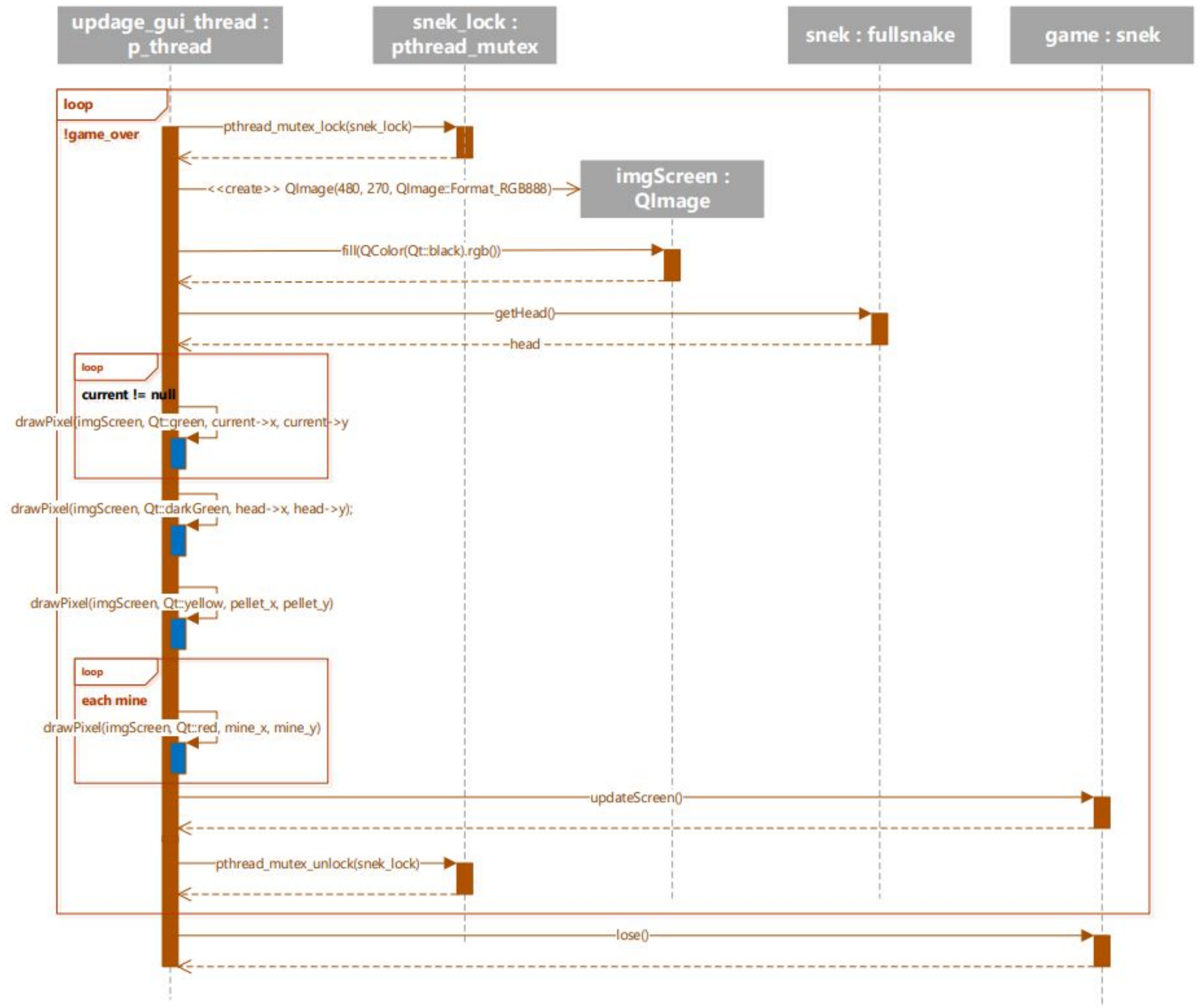
During our previous lab, we discovered we were doing our multithreading wrong. We were using timers instead of incorporating mutexes and semaphores so we had to incorporate those into our design. The first thing that we did was make a datacache class and header to control all of our shared resources. We did this so we would not have to change any of our pre-existing code and could just focus on multithreading. We decided to make semaphores for the snake, pellet, and game screen, as well as mutexes for the snake, pellet, and direction. We used these to restrict each function call so they wouldn't affect any other functionalities, and in return, having a semaphore to refresh the game screen made the game smoother.

2 Multi-Task Design

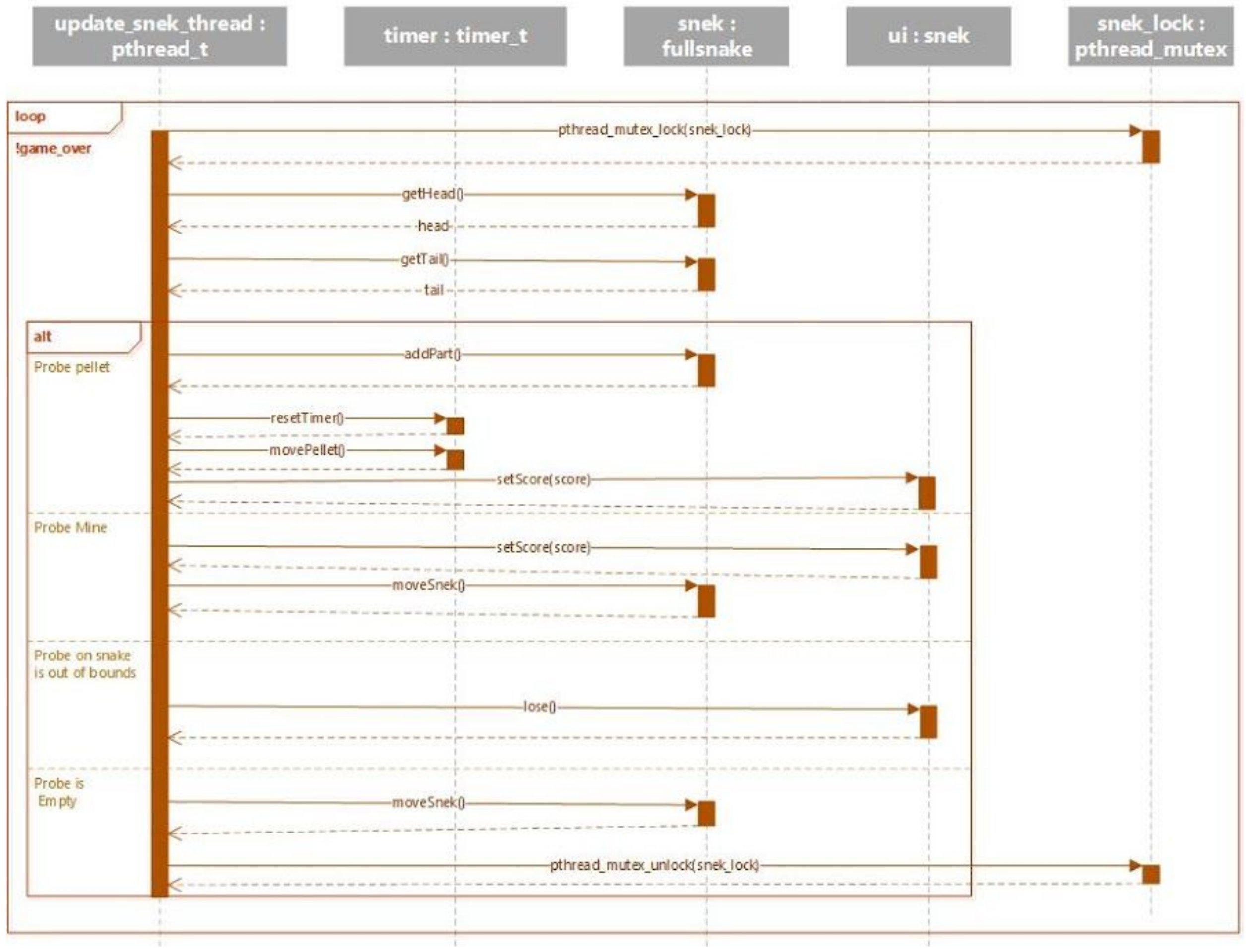
2.1 UML Sequence Diagram

All of these sequence diagrams are to document each scenario of playing the game.

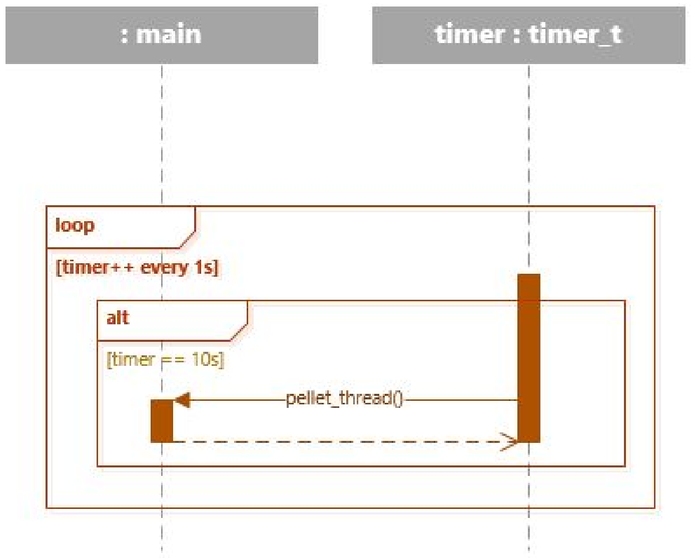
2.1.1 Drawing the Screen


updage_gui_thread : p_thread
snek_lock : pthread_mutex
snek : fullsnake
game : snek
loop
!game_over
pthread_mutex_lock(snek_lock)
<<create>> QImage(480, 270, QImage::Format_RGB888)
imgScreen : QImage
fill(QColor(Qt::black).rgb())
getHead()
head
loop
current != null
drawPixel(imgScreen, Qt::green, current->x, current->y
drawPixel(imgScreen, Qt::darkGreen, head->x, head->y);
drawPixel(imgScreen, Qt::yellow, pellet_x, pellet_y)
loop
each mine
drawPixel(imgScreen, Qt::red, mine_x, mine_y)
updateScreen()
pthread_mutex_unlock(snek_lock)
lose()

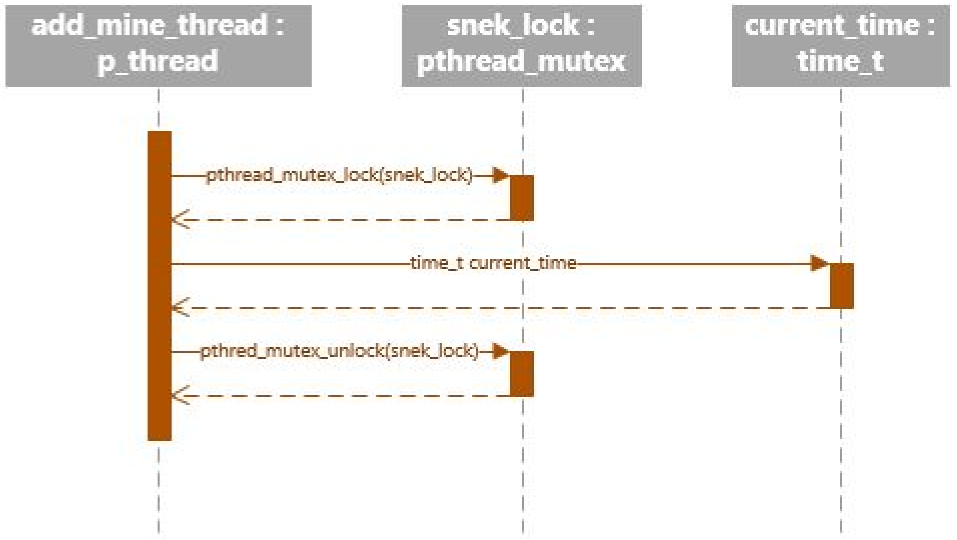
2.1.2 Moving the Snake



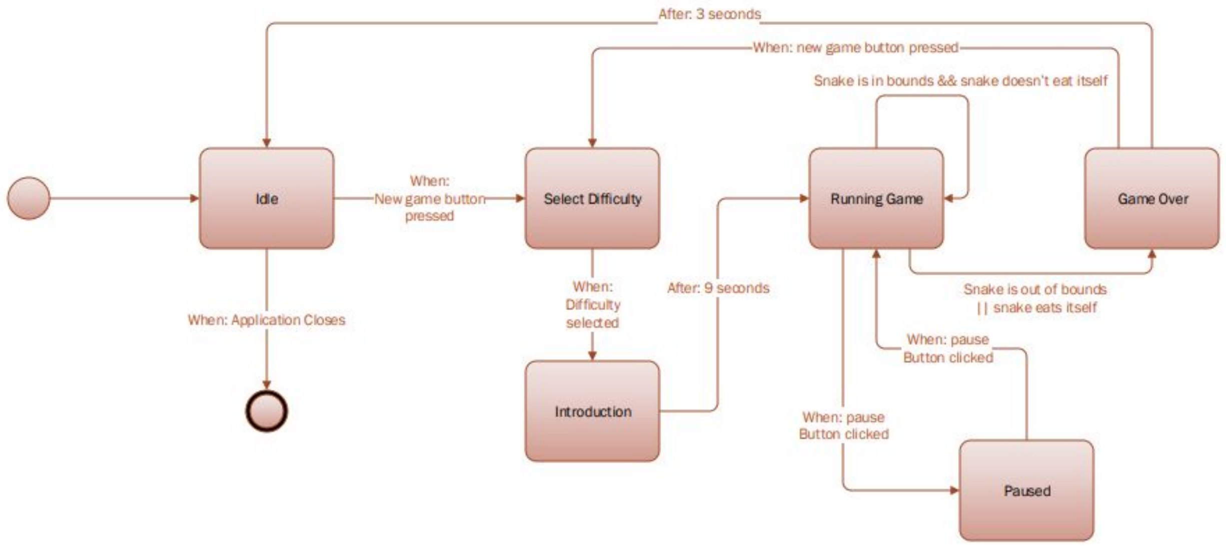
2.1.3 Spawning the Pellet



2.1.4 Spawning the Mine

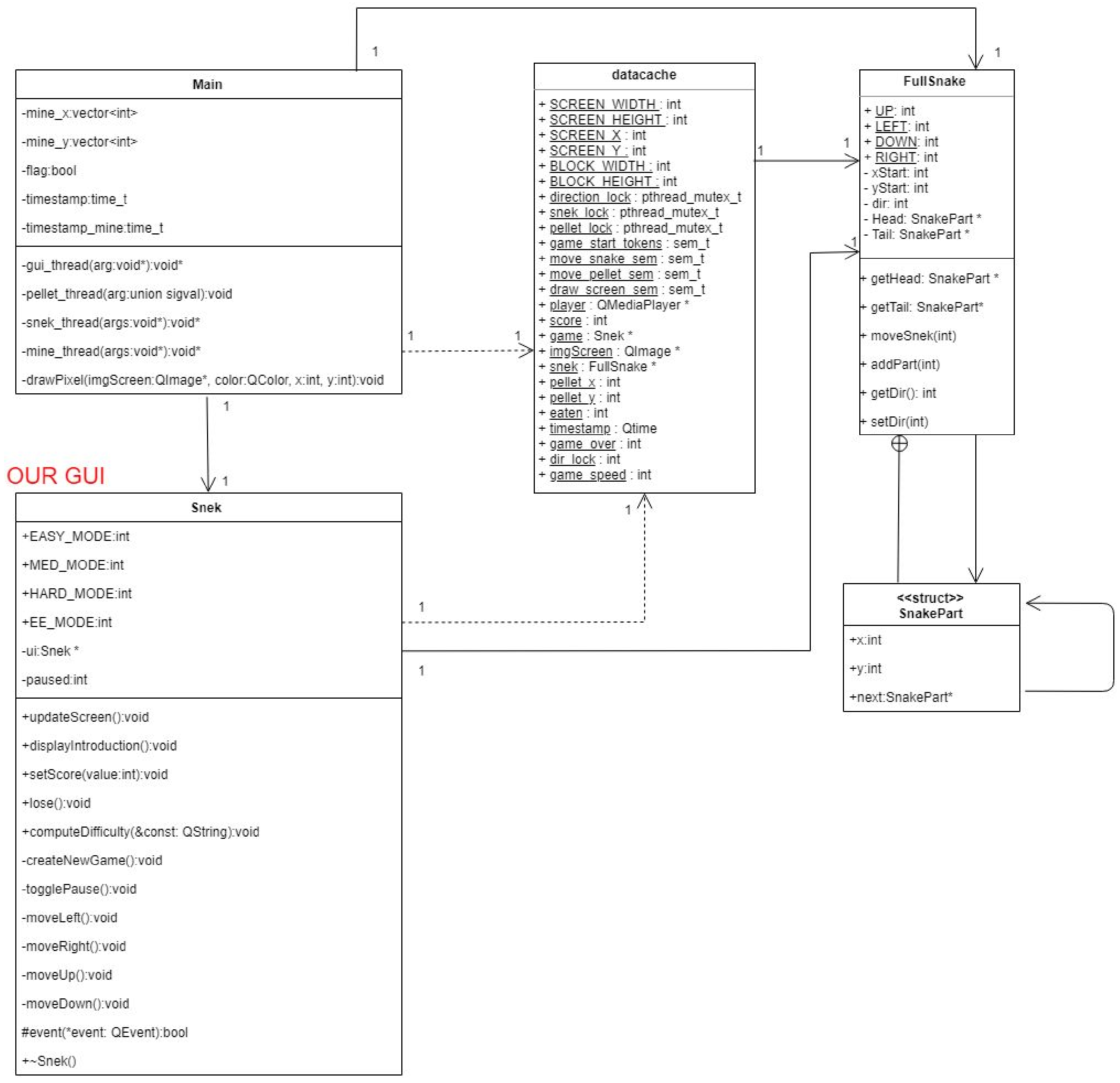

add_mine_thread : p_thread
snek_lock : pthread_mutex
current_time : time_t
pthread_mutex_lock(snek_lock)
time_t current_time
pthred_mutex_unlock(snek_lock)

2.1.5 Game Engine (State Diagram)


After: 3 seconds
When: new game button pressed
Snake is in bounds && snake doesn't eat itself
Idle
When: New game button pressed
Select Difficulty
Running Game
Game Over
When: Application Closes
When: Difficulty selected
After: 9 seconds
Snake is out of bounds || snake eats itself
When: pause Button clicked
Introduction
When: pause Button clicked
Paused

2.2 UML Relations

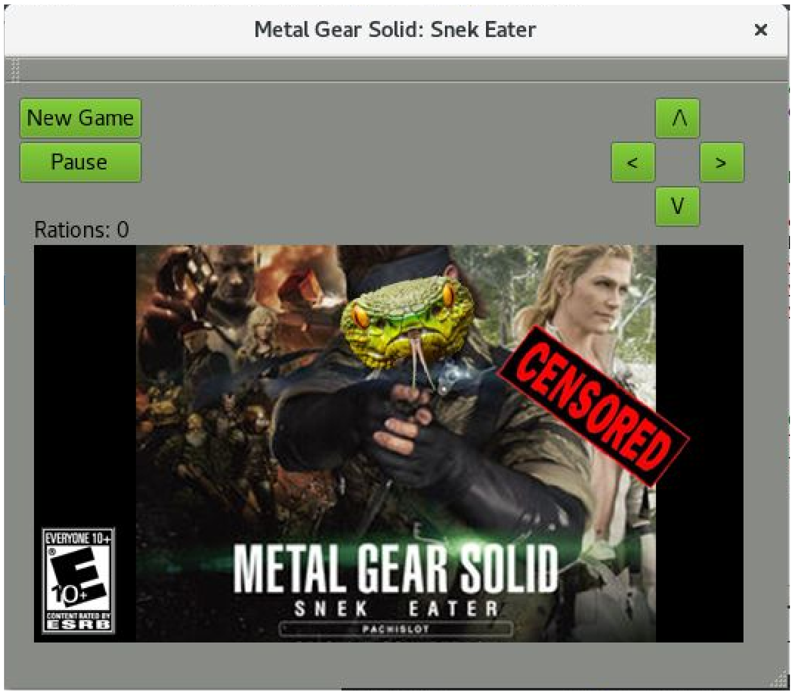
This Diagram shows the relation between the classes in our Project



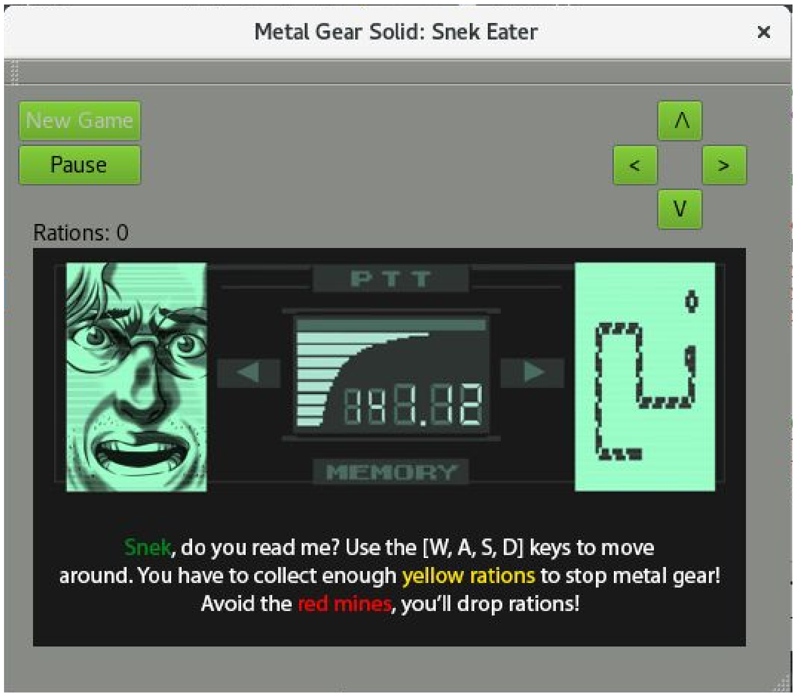
3 System Implementation

3.1 Screenshots of Implementation

MAIN SCREEN:

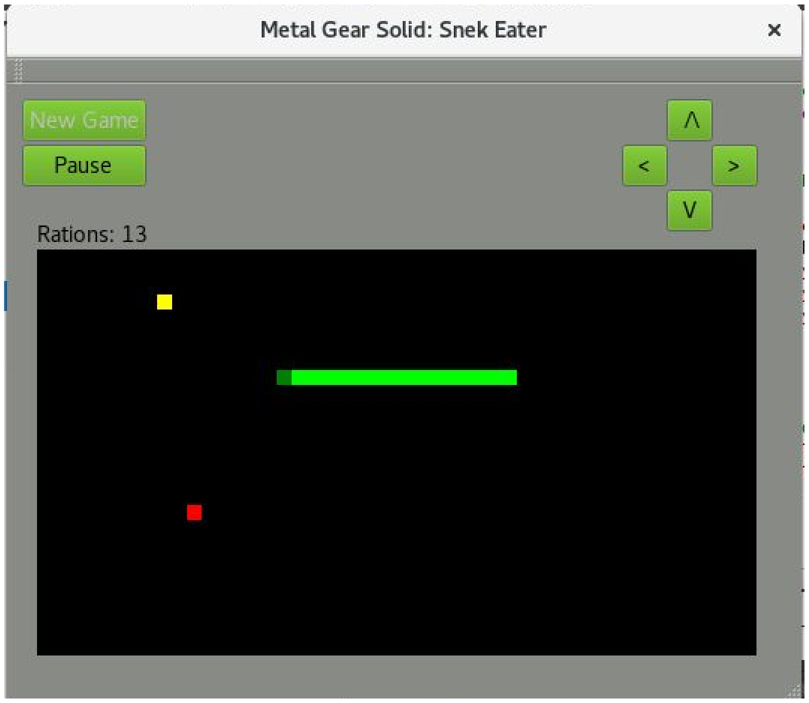


CUTSCENE:

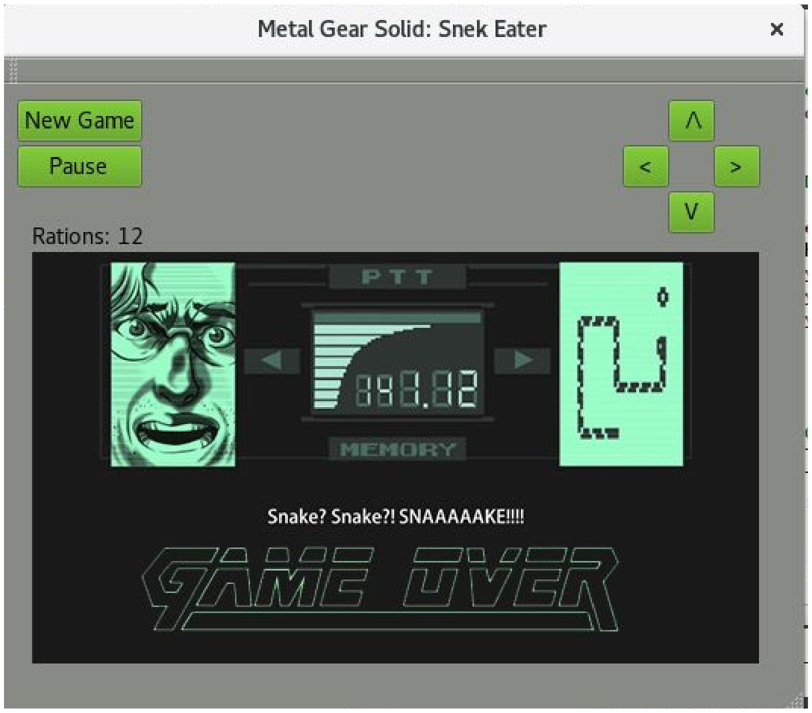


Snek, do you read me? Use the [W, A, S, D] keys to move around. You have to collect enough yellow rations to stop metal gear! Avoid the red mines, you'll drop rations!

GAMEPLAY:



GAME OVER:



3.2 Step-by-Step Instructions

If you are running our project on the beaglebone, you MUST use the no sound version or it will not compile! This is because the beagle bone does not support audio.

1. The first step for building our system is to load up the project onto the beaglebone. This is done by opening up your project directory, select all the files, right click, copy and paste it into the Home folder on the BeagleBone.

2. Next you need to navigate to the file path where you put your project.

3. Once you are there, it is now time to run the command. For example, since the name of our file is "Snek3_SearchForSnek2.pro", we would run the following commands:

   a. touch Snek3_SearchForSnek2.pro
   b. qmake
   c. make

   After a few minutes, your project should then be compiled, and your executeable file will then be called the file name of the .pro file, in our case, "Snek3_SearchForSnek2".

4. After that you need to run the binary file by typing ./Snek3_SearchForSnek2

5. Then your game will run and you can have fun!

4 Lessons learned From The Project

4.1 Dan's Response

One lesson I learned from this project was how to be an effective leader. I took on the leadership role at the beginning and I feel like I have been delegating tasks properly to my teammates. We have a system down where we all go into the lab every week and we split up and each do our own portion of code and morph them so we are all involved and get it done quickly. Being able to do this makes me confident that if I became a project leader in the workforce I could properly manage a team because I have the personality and technical skills.

4.2 Tyler's Response

Coming into this class with no prerequisites, I learned many crucial and fundamental theories behind embedded system design. I learned that Pthreads, when in junction with semaphores and mutexes, are very useful when developing embedded projects. This was shown through our use of threads to control the snake movement and mine. We used semaphores to protect resources between these threads, such as timers and coordinates, which taught me how a system allocates resources within physical code.

4.2 Sam's Response

I learned about the importance of using pthreads over timers. While timers are a way to get something done, it's not necessarily the best. It does not really work well when dealing with multiple systems that interact with one another. For example, with the game screen, pellets, and snake, how the threads run are very important because they affect one another. You have to be able to singal and jump back and forth between them in order to have a game work properly.

4.2 Jonah's Response

This project has taught me a lot about how difficult it is to work with uncooperative embedded systems. The trials and tribulations of working with the Beaglebone and the QtCreator environment has made me wary of working with embedded systems in the future.