REPORT
Daniel Kupisinski

MANUAL:

1. Run the executable
2. Give it a moment to load textures
3. Choose Difficulty Level (Easy by default)
4-SETTINGS is an empty option that does nothing. If chosen, press ESCAPE to come back to MAIN MENU
4. Start Game

This program is very machine – dependant. The speed of the snake is different on different computers.
If the snake is moving too fast or too slow, please see the video of how it performs on lab machines in w115.
You can also tweak the variables with //CONFIG comment – the higher the speed the lower the frame distance should be.

5.!!!Important – While in Game Over screen – press ENTER to see previous scores and difficulty levels chosen.

6.While in Score History, Press ESCAPE to come back to MAIN MENU

CLASSES OVERVIEW:

MENU (10 functions)

- loads in all BMP textures for menu
- draws currently active menu screen
- switches between currently selected options and blits appropriate textures

GAME (10 functions)

- sets up the shader
- **s**ets up the camera
- sets up the light
- instantiates **Config, Snake, Platform, Prize** and **Score** classes
- loads in default values
- sets and resets the game
- sets the difficulty level (increases or decreases snake speed)
- draws the game (calls draw platform, draw snake, draw prize, draw score)

- checks for border collision
- checks for snake – prize collision
- checks for snake self collision – game over

PLATFORM (4 functions)

- sets constant values for **platform pos, rot, scale**
- loads matrices to shader for platform
- draws the platform (**VAO Primitive cube**)
-

SNAKE (6 functions)

- sets initial values for snake **segments rot, dir**
- moves the **snake head**
- updates **snake head borders** each frame
- loads matrices to shader for **snake**
- draws **snake head** (**teapot**)
- draws current **snake segment** (collected **VAO Primitives**)
- stores **snake Pos**, **Rot** and **Borders** arrays

PRIZE (5 functions)

- sets up scale values for different **VAO Primitives** to make all of them equal size
- loads matrices to shader for **prize**
- draws a **prize** in a random position on the platform
- puts new **prize** in a new random position and sets up its **scale** and **name**
- updates borders of the **prize**
- stores **prize name, scale** and **Y Pos coordinate** dynamic arrays (to avoid floating or sinking)

SCORE (6 functions)

- Uses a **Text class** from **SDL TTF NGL Demo** to render score text
- Creates 4 instances of **Text** to render:

  - **current score**
  - **game over message**
  - **final score**
  - **score history**

- updates and draws the **score** when the prize is eaten

- draws **game over screen**
- if requested, draws **score history**

ALGORITHMS :

MENU

I placed my main (**windowOpened**) loop in **main.cpp**. This loop is divided into two Poll Event loops – **Menu** and **Game.** That's because, I use different methods to draw **Menu** and **Game.**

PROBLEM: I had a problem with SDL Blitting textures whenever the game was launched – I couldn't get my **Pause Menu** to display textures. Jon advised me to use **glUseProgram();.** That partially fixed the issue – the texture was displayed, but I couldn't switch between the textures. Later I discovered, that switching **glDepthMask on** and **off** fixes the issue. For **SDL Blitting** – I switched it **off** and for **OpenGL** 3D game – **on.**

MENU MODES: I created a set of **enums** in my **Menu Class** to switch between **Menus** and their **Options** in a **Menu Poll Event** loop in **main.cpp**.

In the end of the **Menu** loop (**windowOpened**) but (**!gameRunning**) I call appropriate functions in **Menu** to draw specific **Menu Screens**.

DIFFICULTY LEVEL: This parameter changes the speed of the snake.
In **GAME** section I will explain why I also change some other variables here.

GAME

In the design stage my algorithm for drawing snake was slightly different. I planned to store snake segments in an array. I wanted my snake game to be smooth and update **snake position** each frame by a small amount. That meant pushing back elements to to arrays each frame.

My logic for drawing snake segments is based on **frameCount.** Each frame, I add an element to the array to store the positions of the snake head. That means the size of the **m_snakePos** array is equal to the **frameCount.**

I decided to use **std::vector** as I needed to control the size of it per frame.

I also have **m_snakeSegments** variable that is initialized to **0 –** meaning, the game starts with just head. That variable gets incremented whenever snake catches a prize.

In my **drawGame()** functon I iterate through all the elements and draw the head - which **position**

is the last element of the m_**snakePos** array, and then I draw other elements which position I define as follows:

I use a variable **m_segmentDistance** which is **m_currentSnakeSegment * m_frameDistance** where:

currentSnakeSegment is the current iteration (**eg,** 0 for head)

m_frameDistance is a fixed value of frames.

Basically, the algorithm draws each segment with a position of the head **x** frames ago.

So, for the 5th segment, the position is going to be the position of the head (last element in the **m_snakePos** array) – **m_segmentDistance**. Hence:

m_snake→m_snakeSegmentPos = m_snake→m_snakePos[m_frameCount – m_segmentDistance];

EAT PRIZE EVENT: I decided to think of my own collision detection method to check for this event. I created two **ngl::Vec4** variables to store 4 borders of the **snake head** and the **prize** in respect to **X** and **Z** coordinates. If one is within each other, the collision happens.

PBOBLEM: I realized that this way, **collision = true** for all the frames in which the snake "passess through" the prize area. To fix that I created a counter **m_timesPrizePut** to make the collision happen only once per prize. That fixed the problem.

PLATFORM BORDER COLLISION: Here, again, I used my own collision detection algorithm. I check if the **snake head pos** exceeds the platform boundaries (which I set manually) and if so :

snake head position = opposite platform border

PROBLEM: I made a logical error here. After assigning snake head position to the other platform border, the collision test will remain true, just for the opposite border – which means the snake is trapped. I fixed that, by adding or taking away **0.0001f** value.

SNAKE SELF COLLISION – GAME OVER TEST: This time, it was far better to use the method I got taught here. I subtract **current snake segment position** from **snake head** position vector and get a length of a new distance vector. Then, I check if the length is smaller than some value, which I had to set up (in this case 0.05 worked fine). That worked perfectly fine.

SPAWNING NEW PRIZE: I use **ngl::Random** to get a random positive integer from 0 to 8.

Based on that, a random **VAO Primitive** is chosen and spawned in a position defined by a random float with range 0.8f;

I also store **prize names** in an **array** to be able to draw the snake as a queue of **VAO Primitives** collected.

I also store **scales** of them and **Y pos** coordinates (to avoid floating or sinking).

SCORE TEXT RENDERING AND CONFIG FILE

I use SDL TTF for text rendering. I downloaded a TTF font "Xcelsion.ttf". I also included the **Text class,** as I was using SDL and could not use **Qfont**.

My **Config** class is responsible for writing and reading from a config file storing the history of scores. I use std::ofstream and **std**::**ifstream** to do that. For **ofstream**, I had to use **std::ios::out |**

std::ios::app flags to allow appending new data to a file instead of truncating it every time I open it. I store:

- Difficulty Level chosen by a player

- Score

I decided to clear the file every time the number of scores hits 8 (the number of scores visible on the screen).

That resulted in not showing the history every 8th time – that's when the history is cleared.

This way, the displayed history is always up do date and showing on the screen.

SHADER: I used a Phong shader used in the demos. To improve the visual impact I use **GOLD** material for **Prize**, **PEWTER** for **Platform** and **CHROME** for **Snake**.

CONCLUSION:

In this project, I focused on learning outcomes. Rather than using provided code, I tried writing my own from **A** to **Z**. I obviously ended up using a lot of code from **NGL Demos** provided by Jon Macey, but whenever I had a change I opted to try to do something on my own, rather than copy pasting. I tried to understand every single line I wrote. For example, I did not use NGLDraw or PaintGL classes, I actually started from the very scratch. My reference for **main.cpp** was **SDLNGL Demo**, but even then, I did not copy paste – I rewrote the lines manually for "muscle memory" - I heard that's a good way to learn coding faster.

Because of this, I made a lot of mistakes and I am aware of them. If given more time I would, definitely fix memory management – I used 6 **std::vectors –** and they had a lot of elements.

To improve performance, I would delete unused elements and even consider implementing a smarter logic, that would not force me to use this many.

In terms of data structure I would also make changes. I would organize my data better and store it in a more structured way.

In my program I had a lot of public variables in classes. I would improve that by creating more getter functions instead.

Also, my program is really dependant on the machine specifications. It runs differently on every single machine. I would improve that, by implementing more stable algorithms (not so heavily based on frame rate).

In terms of a design, it evolved a lot. Back in February my programming knowledge was far worse and I did not predict a lot of things such as data structure, object life-time or memory management. I learned a lot and I enjoyed it – I am looking forward to practice more on my own and enhance my programming skills. Also, unfortunately, in February I didn't submit my design on time, as I had a deadline extension for of my projects (Specialist and VECT). That disadvantaged me from the very beginning but I did my best to keep up. This project, however was very precious to me in terms of learning purely technical skills in computing for animation.