Introduction

Project statement

We implement snake game using MFC and Linked List.

What is snake game?

Snake game is a game genre that first appeared in the 1970s. It is a popular genre in which many variants still appear. In this project, Snake game is implemented using MFC and Linked List which is a data structure. We use MFC to develop the basics of GUI programming. We use LinkedList to implement basic algorithm of snake game, apply data structure to actual problem and develop problem solving ability.

The rules of a typical snake game are as follows.

1. There is a snake controlled by the player in the square where all sides are blocked.

2. The snake does not stop in the direction that the head is heading now. You can only change the direction in which the head moves in the player's operation.

3. A snake dies when it hits its head against a wall or part of its body.

4. The goal is to eat all of the food that lies in a squared-off square. A single food is placed at once without any food, and the food is placed at another location if the food is eaten.

5. Whenever you eat food, the length of the snake becomes longer.

6. Eat all the food, and if the snake takes up the entire screen, the game is over

Project constraint

1. The program is based on a dialog (dialog) among the application types supported by MFC.

2. When we run the program for the first time, a message box asking whether we want to play the game is displayed.

3. If the user presses "No", the program immediately exits and if "Yes" is pressed, the game is executed.

4. The gray rectangle is the area where the snake can move, the range of the area corresponds to 25 x 25 (note that it is not an absolute size on the UI), and it is considered to have hit the wall if it falls outside the area. (In this project, this area is called map.)

5. The green circle is the head of the snake, the yellow square is the body of the snake, and the turquoise circle is the tail of the snake.

6. When playing the game, the head of the snake is located in the center of the map. The body is located in one space to the right of the head, and the tail is located two spaces to the right from the head.

7. the head, body, and tail of the snake should be connected to the Linked List structure.

8. When the game is run, the snake's direction of movement is set to the left, and when the user enters a key, the direction changes The keys to set the direction of movement are 'W', 'A', 'S', 'D' (lowercase letters are also acceptable). If you enter 'W', it will change to the top, 'A' to the left, 'S' to the bottom, and 'D' to the right. At this time, it is impossible to change the direction from the current moving direction to the opposite direction. At this time, it is impossible to change the direction from the current moving direction to the opposite direction.

(example)

1. If the snake is moving upwards, it can not change direction downward.
2. If the snake was moving downward, it can not change direction upwards.
3. If the snake was moving to the left direction, it can not change direction to the right direction.
4. If the snake was moving in the right direction, it can not be changed in the left direction.

9. When outputting the motion of the snake on the screen, the update period is limited to 55ms. This is implemented using the Timer function provided by MFC.

10. The red rectangle is the snake's prey, and when the snake eats it (when the snake's head reaches the area), the length of the snake's body (yellow squares) will increase by one. Thereafter, the food is randomly placed at different locations on the map.

11. There are two cases when the game ends.

    1. The first is a situation where the head of the snake bumps into the wall. In this case, the game is ended by outputting the message box (제목: 실패, 내용: 벽에 쿵!)

    2. The second is a situation where the head of the snake bumps into the body. In this case, the message box (제목: 실패, 내용: 몸에 쿵!)

12. There is no limit to the size of the UI. The area of the area where the snake can go is 25x25, the color and shape of the part of the snake, the color and shape of the food should be the same as proposed in the project, and in some cases the contents of the message box should match.

Project solution

1. The Snake class declaration, which is the Node class that will make up the snake's.

2. Declare Linkedlist class to follow the Snake class.

3. I declare a constructor that will link the snake's head body to the LinkedList class initially.

4. In the LinkedList class, declare AddSnake (int xPre, int yPre) to increment the linked list when the snake feeds.
This function takes the x and y coordinates of the snake's head as arguments.

5. Timer function declaration to perform different actions on keyboard input.
This function change Linkedlist's x and y coordinates correctly into keyboard input direction.

6. Declare Function to handle keyboard input.
This function Recognize 'W','S','A','D' in keyboard into Respectively up,down,left,right

7. Declare Function to check if Snake’s head hit body.

8. Declare Function to check if Snake’s head hit food.

10. Declare function that redefines the location of food through random functions.
This function is declared by a recursive function to prevent situation that the position of the food is in the body of the snake

11. Write code to the Onpaint function to draw walls, movable areas, and snake and prey.

Algorithm and Flow chart

Project algorithm

1. Use MessageBox function in OnInitDialog to ask whether you want to proceed with the game.

2. If you do not want the game to proceed, exit through the if statement, and if you want to proceed, run the OnPaint function.

3. The OnPaint function uses the for and if statements to draw a wall and a movable area where the snake can not move. And draws snakes and food.

4. When the user enters the keyboard, the case statement of the OnKeyDown function determines the corresponding alphabet and executes the SetTimer function.

5. In the SetTimer function, the timer type is divided into case statements. Each timer is executed by the OnKeyDown function.
The SetTimer function finds the head of the snake through the while statement and determines if the head of the snake has hit the wall, body, or food through the if statement.

6. If you hit a wall, exit the while statement, display a failure message, and exit the program.

7. To determine the bump on the body, the SetTimer function executes the BodyHit function.
The algorithm of the BodyHit function is as follows.
Save the moved head position and run the while statement to see if it is the same as the position of the snake stored in the linked list. If there is something like this The program terminates with a failure message.

8. When there is no bump
The SetTimer function moves the head of the linked list and the body tail one space at a time through the while statement.
The algorithm for movement is as follows.
The head is moved first and the body is moved to the empty space. At this time, as the empty space is generated, the body and the tail are moved until the empty space is eliminated.

9. The SetTimer function runs the FoodHit function to determine if it has hit a prey.
The algorithm of the FoodHit function is as follows.
The position of the head of the moved snake is stored separately and compared with the position

of the food. If the positions are the same
Adds a new node by adding the position that the snake's head had before moving it and adds one body by executing AddNode function.
The algorithm of the AddNode function used in this case is as follows.
If you pass the position of the head of the snake that you moved before as an argument to the AddNode function
The AddNode function then dynamically allocates the new node and stores the passed arguments in the x and y variables of the new node.
The next step is to run the FoodPositionChange function to change the position of the food.
The algorithm of this function is as follows.
The x and y coordinates of the food are changed through the random function. But the important point here is that the modified coordinates should not be the same as the coordinates of the body and tail of the snake, so use the while statement to compare the snake's body and tail marks.
At this time, if there is the same coordinates, the function FoodPositionChange is executed again. That is, the function FoodPositionChange is recursively executed in some cases.

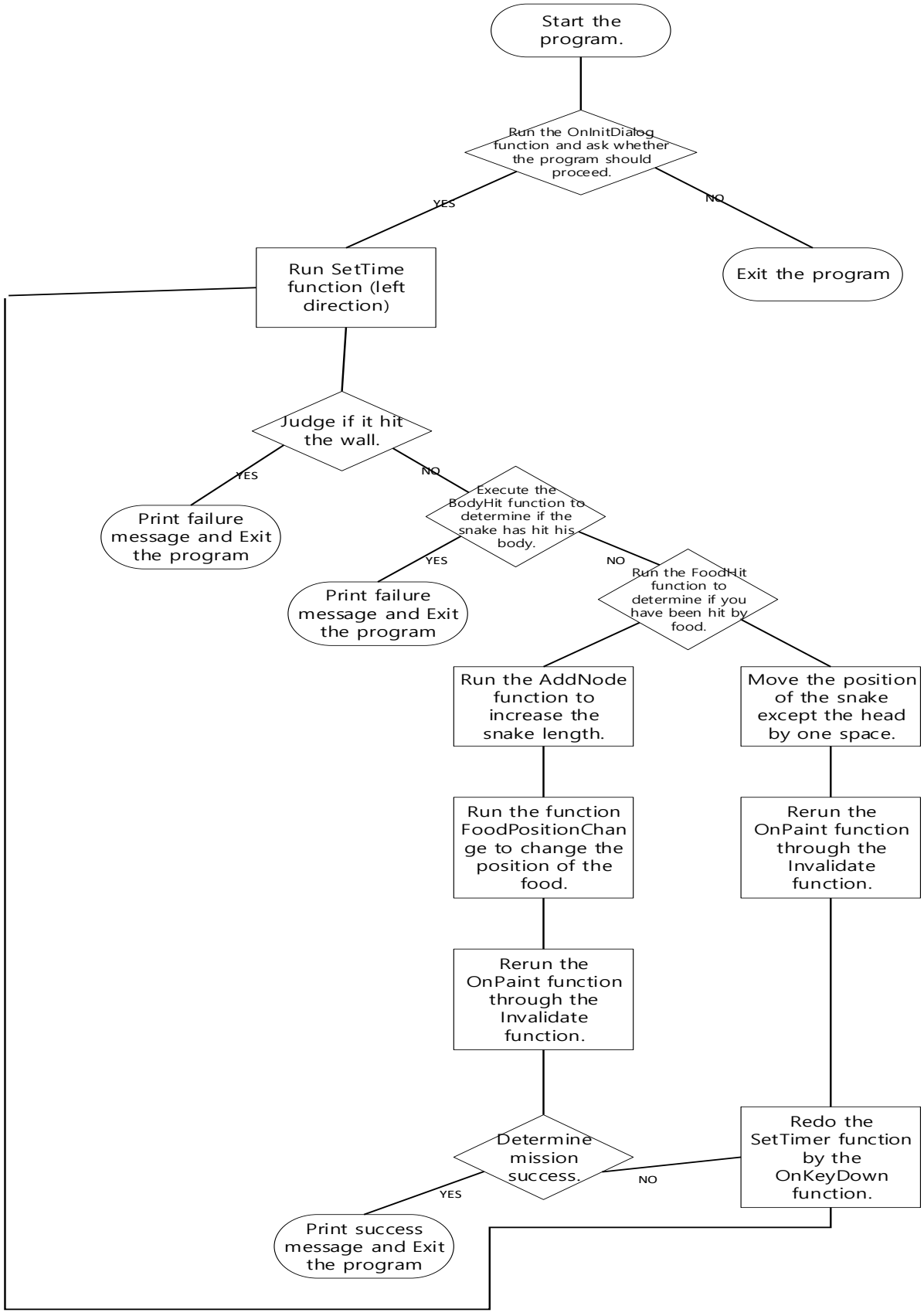
10. The above general procedure is executed consecutively with SetTimer iteration. When the snake hits the wall, it uses the KillTimer function to stop the SetTimer function.

11. Also, when the SetTimer function of another ID is executed by the OnKeyDown function, the previously executed SetTimer function is terminated by using the KillTimer function.

12. In addition, the algorithm to determine the success of the mission is as follows.
Declare FoodEatCount as a global variable and initialize it to 0. If you eat 623 pieces of food based on the field of 25X25, and if you eat 623 pieces of food, print out the message of success with the FoodHit function and terminate the program.
At this time, the FoodHit function is responsible for increasing the FoodEatCount variable.
And why should you eat 623, because the early snake should occupy three of the 625 areas, and the maximum number of feeds must be 623.

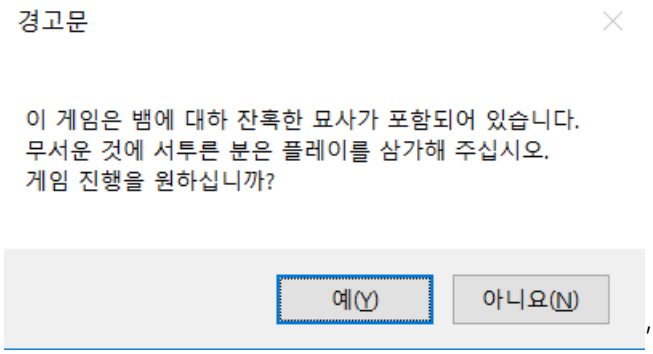
13. The animation effect uses the Invalidate function to re-execute the OnPaint function whenever the position of the snake is changed in the SetTimer function.

Project Flow chart

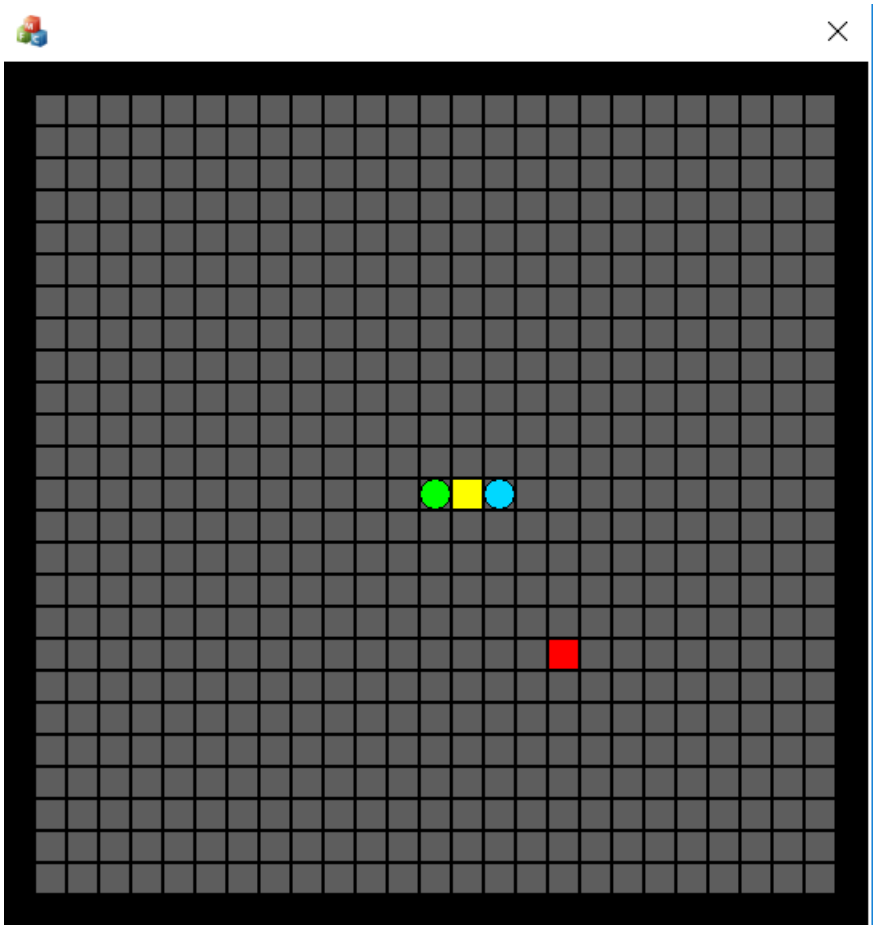


Results screen

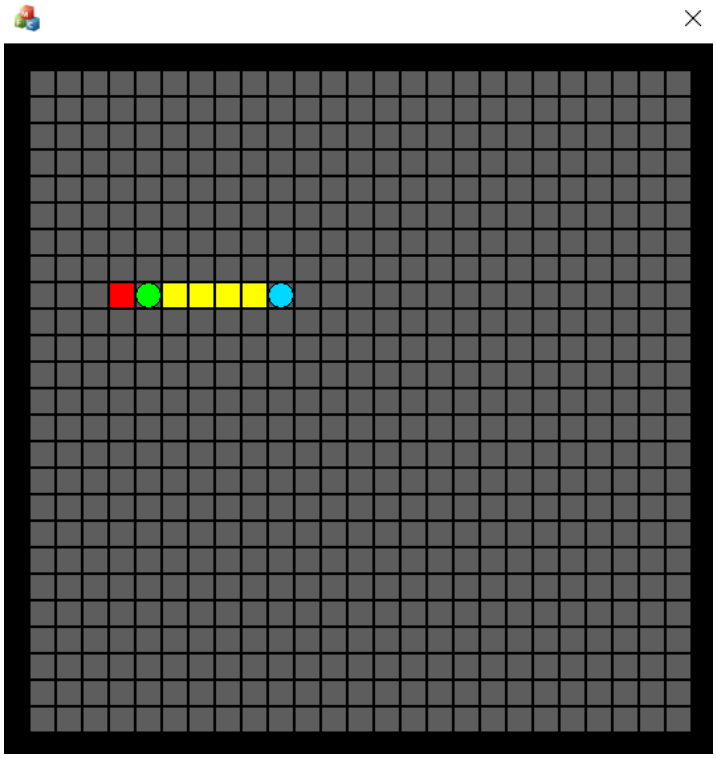
Message box displayed when program is executed



The initial screen displayed when the game is executed

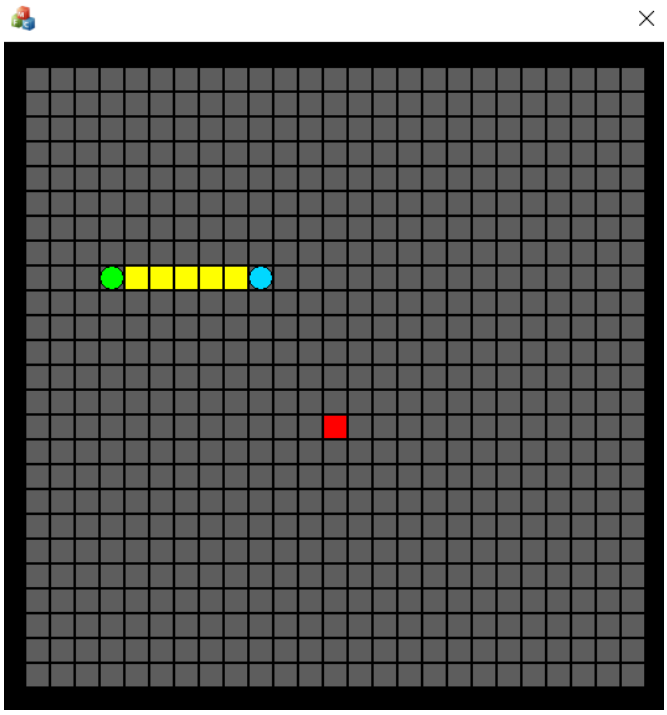


Situation just before eating food

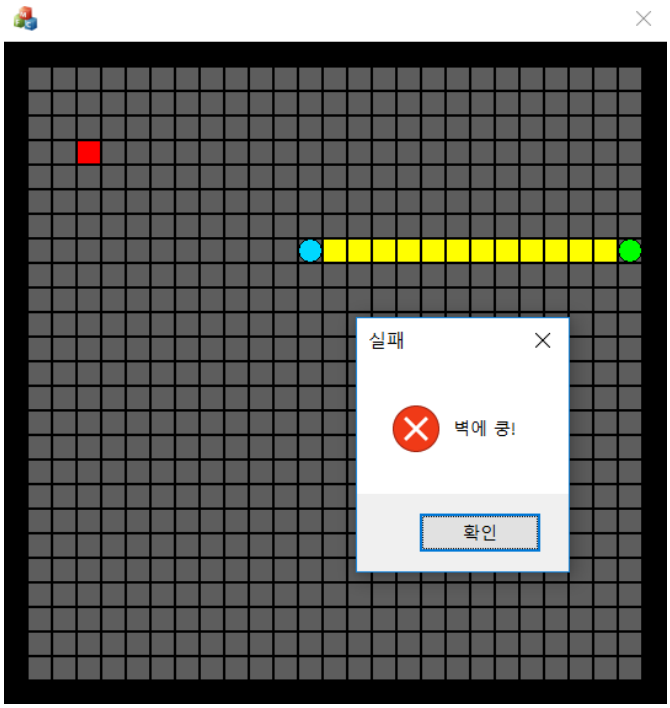


Situation after eating food

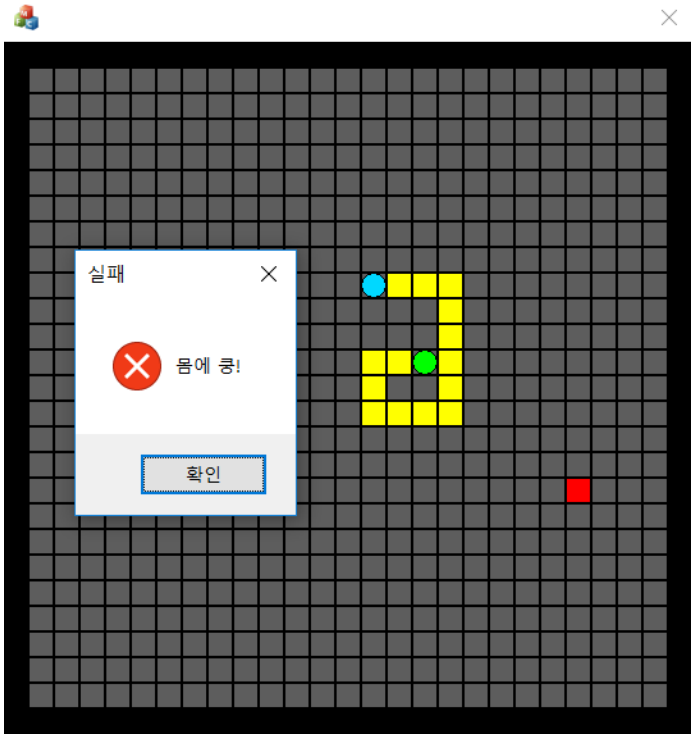
(You can see the increase in the length of the snake and the change of the food position.)



The screen when the snake hit the wall.



The screen when the snake hit the body.



Review

To illustrate the advantages of the above program, we first declared the map's vertical size and horizontal size as #define, so if you want to modify the size of the area, you only need to modify the #define code.

And the FoodPositionChange function was designed to behave as a recursive function in some cases, which prevented prey in the position of the snake. This way, the prey is in a position other than the body until the end of the game.

There was also a problem with implementing the program. The snake, including the prey, was flickering on the screen.

Looking for ways to solve this problem, I found out about the double buffering technique

It was solved by utilizing the memory DC class and the CBITMAP class.

In addition, it was not related to the grade of the project, but it was implemented in case of success.

I can see through the result screen below that the output appears correctly.

(I set the size of the existing 25X25 to 5X5 size to see the output smoothly.)