Snake++ Final Documentation

Group 39

Nader Hosseini Naghavi
Piranavan Jeyakumar
Tianci Du
Wentao Lin
Ziqingqing Ye

Project Summary

Based on the classic game, Snake, the user will be controlling a snake using the keyboard (e.g Up, Left, Down, Right) to move it on the game board and off screen. Additionally, “fruits” and “bombs” will be randomly generated on the screen, the fruits serve to increase the player’s score, but also the size of their snake, whereas if the snake collides with a bomb or itself, the game will be terminated. Every time the snake eats a fruit, a new fruit will be randomly generated to replace it, and the locations of the bombs will randomly change whenever the user’s score is a multiple of 5. The locations of the fruits and bombs go through an algorithm to ensure that the objects do not overlap and that the bombs are not generated too close to the head of the snake (to ensure the game doesn’t end unfairly).

The goal of the game is to get the highest possible score, without having the snake collide with any of the bombs or itself. Upon termination of the game, a high score board will be shown, containing the top 5 scores of any past games, and will automatically add the user’s score if it surpasses any of these 5 scores.

Key Accomplishments

What went right?

Our development of the application on Qt worked very well, as we included a variety of Qt objects, many of which made it easier to implement specific features in our game. For example, the use of QPushButton to start, pause, and resume the game. Additionally, our use of the Timer function and KeyPressEvent function helped make the game much more interactive and smooth to play, as there were rarely delays between pressing the key and the actual movement of the snake.

The generation of the bombs and fruits, and using the algorithm to determine whether their locations were a fair distance from the position of the snake's head also worked correctly. We were able to generate these objects randomly and ensure they follow a certain set of rules to keep the game difficult, but not unfairly so.

The implementation of the background music worked well. When playing the snake game, there will always be background music playing. Additionally, whenever the snake eats the fruit, there will be an eating sound, just like the snake is actually eating the fruit. When the snake collides with a bomb, there will also be an explosion sound so that the user can be notified of the end of the game.

The collision effect and scoreboard functions also worked correctly. When snake collides with a bomb or the snake body, there will be a collision effect shown on the screen so that the player can be aware of the location of the collision. In addition, players can see the current score and the top 5 scores are shown in order in the scoreboard. Furthermore, the past scores will be stored in a file and are compared with the new score to determine if the new score must be saved.

One other feature of the game that was developed and worked fine, was the use of 1-D arrays and vectors to store the location of Snake, Fruit, and Bomb objects. Although we faced some challenges in using 1-D data structures to store 2-D coordinates of pixels, we managed to use 1-D data structures and this method made other processes much easier. For example, by having 1-D vectors of snake positions, we could easily implement the snake growth and movement, random regeneration of fruits and bombs methods. That is because the interaction with 1-D data structures (like removing a data point or regenerating random data points in 1-D vectors) are simple.

What was found to be particularly useful?

There were many aspects of our project that were useful and made the implementation of specific features much simpler. A notable aspect was the use of the 1-D array to contain the information of each of the snake's body positions on the board (the snakeBoard variable). This 1-D array made the implementation of the snake movement trivial, as to find the location of the where the snake's head should go to, you would only need to determine the snake's direction (stored in a string variable that changed value based off last key pressed) and add an integer to the index to determine the index of the new location.

Additionally, the use of the QtCreator IDE helped make the development of the project go smoothly as it would auto-suggest libraries we would need for the Qt objects we were implementing. The IDE would only show the .h, .cpp, and additional files only if they were included in the game.pro file, ensuring that we update it if we added any new files. The IDE also made the execution of our project extremely easy, as we would need to click the "RUN" button, as opposed to doing qmake, make, and then running the executable, on the CS3307 remote server.

The use of the Qt framework helped make our project work on different operating systems, which increased the portability of our project. Since it was cross-platform, our team members were able to test the project on their own operating systems (Windows, Mac, Linux), without having to deal with OS specific errors.

Which design decisions contributed to the success of the project?

A design decision that made the implementation of certain features like snake movement and collision detection a lot easier to do, was using vectors to store the snake's positions, fruits' positions, and bombs' positions. Using the std vectors allowed us to access pre-made iterator and access functions, allowing us to modify and add new elements to the vectors with ease.

The use of a 1-D board to hold information about the game board was another decision that made displaying the contents of the board much easier. Instead of creating the object (snake_body, fruit, or bomb) and then painting it and adding it to the board, all we would need to do was to say that this object would be present at a specific point at the board (decided by our placement algorithm) and then the board display would be updated (everytime the Timer class ticks). This made displaying the fruit, bomb, and snake objects much easier as all we needed to do was to add it to the board and then add a corresponding colour for the object in the code, and it would be displayed immediately after.

The movement of the snake also contributed to the success of the project. Instead of monitoring each point of the snake's body and changing their locations accordingly, we simply monitored the head and tail of the snake. We would remove the tail of the snake from the vector and add a new element, which would have the predicted location of the snake's head given the direction, to the vector.

Another important design decision that contributed significantly to the success of the project, was our decision to mainly use the QT library for the user interface of our game. This library comes handy in the development of graphical user interfaces in c++. That is because this library provided us with different graphical objects we needed including a frame for the main game object, push buttons used for starting and pausing the game, and labels and LCDs used for showing the score and other information.

Another decision strategy that made the development of different components of our graphical user interface simple and efficient, was using two different classes: one for the main board of the game showing the snake, fruit and bomb objects, and another one for the main user interface of the game including the frame on which the game is shown and other UI components

like the LCD, push buttons, and labels. This design strategy was mainly useful since we could develop one independant class that renders the main game frame including the board of the game, snake, fruit, and bomb ojects. Then we could use the class used for the whole user interface of the game to integrate the mainwindow object (object controlling the main frame of the game) with other components in one screen using the QWidget method of the QT library. In addition, we could easily establish the connection between other UI components of the game like push buttons, LCD and the object controlling the main game frame in the class controlling the whole UI of the game.

Key Problem Areas

What went wrong?

One of the things that went wrong was that at first we wanted to have some animated shapes for our snake object, fruits, and bombs but since we mainly used QT library for the main interface of the game, we could not find a method for showing animated figures instead of colored pixels in the main frame of the game. We mainly used the Painter object of the QT to draw the pixels with different colors on the main board and we were not able to adjust it to draw a snake-animated figure.

Another thing that went wrong was getting the background music to work on the remote server. Playing the background music requires the Multimedia package in the QT library, and since we did not have the permission to install this package in the CS3307 server, we could not test the final version on the server. Instead we tested it in the QT Creator IDE on our own computers.

One other issue was that in our initial plan, we wanted only one board to be used in the game to show the main interface of the game comprising snake, fruits, and bombs objects. However, we realized that in addition to the main board of the mainwindow class, we also needed a similar board for our snake, fruit, and bomb classes to only hold those shapes; then we combined all of these boards in the mainwindow class.

One more thing that went wrong was one of our optional features, we wanted to use hand gestures or hand movements to control the snake's movement by using the webcam. However, it is able to detect the finger movement but sometimes messes up with other stuff like face or anything appearing in the background. This might be related to webcam quality or our code still has a very large improvement space. Eventually this optional feature failed due to lack of time.

What project processes didn't work well?

Overall, the majority of our project processes worked well, our planning and completion in particular went smoothly and was done in an efficient manner. Our project implementation on the other hand could have been done better. The main issue with our project implementation was that we deviated from the UML diagram created during the project planning phase. Due to this, we needed to completely restructure the UML diagram to ensure it made sense and that the interaction between the different classes were meaningful. Although our code worked nicely on run-time, it would've been nicer if we had organized our code into their proper classes, to easily show how the different classes interacted with each other.

What technical challenges were encountered?

One of the technical challenges that we encountered was using the QT Painter object to draw the pixels on the main window of the game. One problem was that we did not know how to call the QT Painter event on the QT Timer class such that the board pixels would be updated on

each tick of our Timer class. Later we learned that by calling the update() method, it would call all of the needed Painter events to update the board.

The other technical challenge was that we had a problem with drawing each pixel of the screen with the color corresponding to the object on that pixel of the board (NoShape, snake body, fruit, or bomb). Later we realized that we should use the Painter object to draw a square and fill it with the related color using the pen attribute of the Painter object.

Another technical challenge, was to determine how to define the data structure of the main board of the game (and the boards used in Snake, Fruit, and Bomb classes). Initially we tried using a 2d array for representing these boards, but then we found it difficult to modify such a data structure. Therefore, we used a 1-D array to represent the main boards as it made implementing functions like snakeMove() easier and it allowed us to easily modify the board.

The other challenge was how to use a 1-D array to represent a 2d grid. Since the game board is a 2d structure, how would we use a 1-D data structure to represent it? Usually in similar applications 2d arrays are used in C++ to handle a 2d grid; however, we found this data structure to be quite inefficient because of two reasons. One reason was that updating these data structures would be more time consuming than a 1-D array. In addition, we also needed vectors for storing the specific locations of each snake, fruit, and bomb objects. Since 1-D vectors are more efficient and easier to use than 2d vectors, we decided to use 1-D vectors instead. The method we used for solving this issue was circular encoding. In this way, the 1-D coordinate of each pixel is its x coordinate added with the length of each row multiplied by the number of lines below this pixel.

Another challenge we faced was making sure our program worked on all operating systems (to allow other group members to run it on their own computer). In our initial discussion, we did not limit the use of C++ libraries. For the random generation of fruits and bombs, we started by using C++ random library which works fine on the macOS system, but did not work on a Windows OS.

<u>What design decisions made it more difficult to succeed in your project?</u>

One of the design decisions that made it more difficult to proceed in the project was our decision to create each class (mainwindow, Snake, Fruit, and Bomb classes) with its own specific board. The problem was that we had four different boards for our main frame. snake, bomb and fruit objects, we should have found a way to integrate all these boards together, instead of having 4 separate boards.

Another decision that made things a bit difficult was to implement our optional features, playing background music and sound effects (for snake eating and snake collision). Since background music files are normally larger than sound effects, we needed to choose different file formats to serve as background music or sound effects. Besides that, there were issues in finding the related file in our project. At first, we put music resources in a folder and used the relative path to find those files. However, even though there were no error messages being displayed, the music would occasionally not play.

What were the effects/impact of these problem areas?

One of the areas that was affected was the accuracy of the game (i.e. does the game work as it should?) due to having separate boards in our mainwindow, Snake, Fruit, and Bomb classes. The issue was that if we failed to accurately integrate these different boards, the shapes might have been shown at incorrect locations or two different shapes may have overlapped with each other in the main frame.

Another issue that affected accuracy of the game, was using 1-D vectors and arrays to represent the location of shapes on the frame. If we failed to accurately transform 2-D coordinates into 1-D coordinates using circular encoding, the shapes might have been shown in incorrect positions or might have even been shown outside of the screen.

What corrective actions did you take to resolve the problems?

In terms of the issue with using animated figures to show the objects on the screen including snake, fruits and bombs, we mainly used drawing pixel utility of the Painter method of the QT library, but we used informative colors for those objects so the user can easily identify them. For example, we used black for the snake's body, green for each fruit object, and red for each bomb object.

For the issue we had by having separate boards in classes mainwindow, Snake, Fruit and Bomb classes, we used getter methods in Snake, Fruit, and Bomb classses to return their specific board, and in the mainwindow class we developed a function to integrate all of these boards with the main board of the game.

The problem we had with drawing pixels with their specific colors on the main frame of the game, we realized that we should use the Painter object to draw a square and fill it with the related color using the pen attribute of the Painter object.

In order to resolve the issue related to using 1-D vector and array for holding the position of shapes (snake, fruits, and bombs) on the screen, we used circular encoding. In this way, the 1-D coordinate of each pixel is its x coordinate (in 2-D space) added with the length of each row multiplied by the number of lines below this pixel.

To make our project work on different platforms, we used the QtCreator IDE since it supported multiple platforms (i.e. cross-platform), and we would not encounter any issues with certain libraries on specific platforms.

In terms of the problem occurring when adding the background music, we decide to use the .mp3 format as our background music, and the .wav format as sound effects for the snake eating a fruit and the snake colliding with a bomb. We then decided to put music files inside the resources folder under Qt and this fixed the issue where the music would occasionally not play (even though no error messages were displayed).

Lessons Learned

A lesson we learned was to precisely make plans about the methods we want to have in each class and its structure and strictly follow those plans throughout the development. One of our problems was deciding whether to have some game features like snake movement and growth in the Snake class or in the mainwindow class, which caused some confusion for us and delayed progress on the project.

Another lesson learned was to consider the problem of operating cross-platform. Especially for group projects, unexpected errors coming from different operating systems would be annoying. Hence, using a framework that works cross-platform would solve this issue and make group progress go smoothly. Originally, we did not discuss the use of IDEs and instead used whichever IDE we were most comfortable with. Later on, this caused a few issues/errors and the project was delayed a bit to resolve these issues. After that, it was decided that we would all use the QTCreator IDE on our local machines, to reduce the number of issues occurring because of different Operating Systems and because since our application used QT, using it's corresponding IDE helped a lot with understanding Qt and knowing which libraries to include.

Going forward, we should implement our functions and classes following the UML diagrams we created because once we deviate from our blueprint, our project structure begins to collapse and it becomes difficult to understand how different components interact with each other. I believe this issue could be resolved by having more group meetings and having better communication with the entire group to ensure we are all on the same page and that we all understand and agree with the layout.

Wentao -

I would love to work on a similar project again in the future as working in a group is quite different from working individualy. It is definitely a great experience to gather a group of people together and kick around a few ideas. We discussed and brainstormed. When we encounter some issues there is always one person who comes up with a great idea and then we keep optimizing it to make it perfect. Also, I learnt that before submitting my code we should always make sure our own code runs smoothly and clean so it will not cause any problems or conflicts with other teammates. Overall it is very nice to see the whole project is running as expected.

Nader -

I would definitely like to work on such projects in the future since this project was very useful for me in developing some skills. First, in this project I practiced team working strategies like planning the development of the project in multiple steps, and dividing the work in each step into independent components that each individual member can work on, and communicating with other team members during the development to stay on track. I found this strategy useful in

development of software packages as a team and I am definitely interested in using such strategies in the near future.

Second, a technical aspect of the project that was very interesting for me was to learn how to develop efficient and integrated User Interfaces for software packages. Before the development of this project and the individual assignment of the course, I did not have much experience in designing the UI components in C++, but I found it very useful and interesting. Specifically, I liked the usage of the QT library for managing different UI components and how we can employ Object Oriented programming strategies to render the whole system. OO methods can help us to use independent objects for rendering different UI components and using a main object to connect them all together in the main UI of the software package. Overall, I really liked the Object Oriented design methods and UI development and I would like to work on similar projects in the future.

Piranavan -

I would like to work on a similar project in the future as I do not have much experience working in groups on a project where we almost have complete control of what we would like to do. I believe this experience helped me understand what I should do better to ensure the group is communicating properly and to ensure the group sticks to the plan that we initially agreed on. I also think I would want to have 2 people working on implementing any specific feature to ensure the work gets done, and on time.

Tianci -

I would like to work on a similar project in the future. In this project, I have discovered my deficiency in group working. So, I expect more chances of group working so that I can improve my teamwork ability. Different from individual work, knowing how to code does not mean success in group working. Without effective communication, group work might be even less efficient than an individual job. As we need to make sure the code we generated can be run in other teammates' computers, as well as understanding how other teammates work so as to interact with their code. Besides, I also learned that checking the team progress frequently is necessary as it is a way to achieve our initial goal as much as possible.

Ziqingqing -

I would like to work on a similar project in the future because I would like to learn how to work productively with others. Most of the time, I am so used to working individually but this time I feel more motivated when I work with a group. I enjoy having other people around to bounce ideas. With properly structured, group work reinforces skills that are relevant to both group and individual work, including the ability to refine understanding through discussion and explanation and develop stronger communication skills. I also learned that I should frequently test and verify the code at each stage. Require testing before any pull requests to an important branch. This is especially important to prevent merge conflicts. Additionally, when we work as a group, the code should also be well-organized and easy for other team members to understand.