Brawl Snake

Final Report

Binary Brawlers

Cooper Braun, Navin Kunakornvanich, Brady Russell, Francesca Strickland-Anderson

Course: CPSC 224 - Software Development

I. Introduction and Project Description

This document summarizes our project's planning, progress, and technical details. For our version of Snake, we based our rules on the original Snake game and added visual and audio inspiration from *Minecraft* and *Pokemon,* and our group and game name are inspired by *Brawl Stars*. (add to appendix) The game aims to eat food to grow and score as many points as possible without colliding with obstacles, the wall, or yourself. The user can use WASD or arrow keys to move the snake on the board. As for food, the red apple grows the snake by one segment, the golden apple grows by five segments, and the star fruit grants invincibility for 12 ticks.

II. Team Members - Bios and Project Roles

Francesca Strickland-Anderson is a Business Administration student with a minor in computer science interested in software development and international business. Her prior projects include creating a family-run fundraiser to raise money for children in orphanages. Francesca's skills include C++, Python, HTML, SQL, and Java. For this project, her responsibilities include creating a timer, adding the background and music, finding images and audio, and leading for nontechnical work.

Cooper Braun is a Computer Science student with a concentration in data science interested in machine learning and data wrangling with a passion for mathematics. His prior projects include a web development cheat sheet to help beginners with full-stack programming. Cooper's skills include C++, HTML, CSS, JS, SQL, Java, Python, and other libraries and frameworks like Pandas, NumPy, Matplotlib, and React. For this project his responsibilities included implementing obstacles, the different foods and their properties, key listeners for user input, continuous snake movement, some of the overall game logic, and filling in information for most of the slides.

Navin Kunakornvanich is a B.S. Computer Science student with a concentration in software app development. His prior projects include creating a fun and interactive online survey with HTML, CSS, and JavaScript through React on Visual Studios Code. Navin's skills include C++, Java, HTML, CSS, React, Swift, NetLogo, Python, TypeScript, Angular, and Node.js. For this project, his responsibilities included frontend development designing the GUI, creating individual windows such as the start menu and game over menu, and linking the logic from the backend Java code to the frontend JavaSwing code.

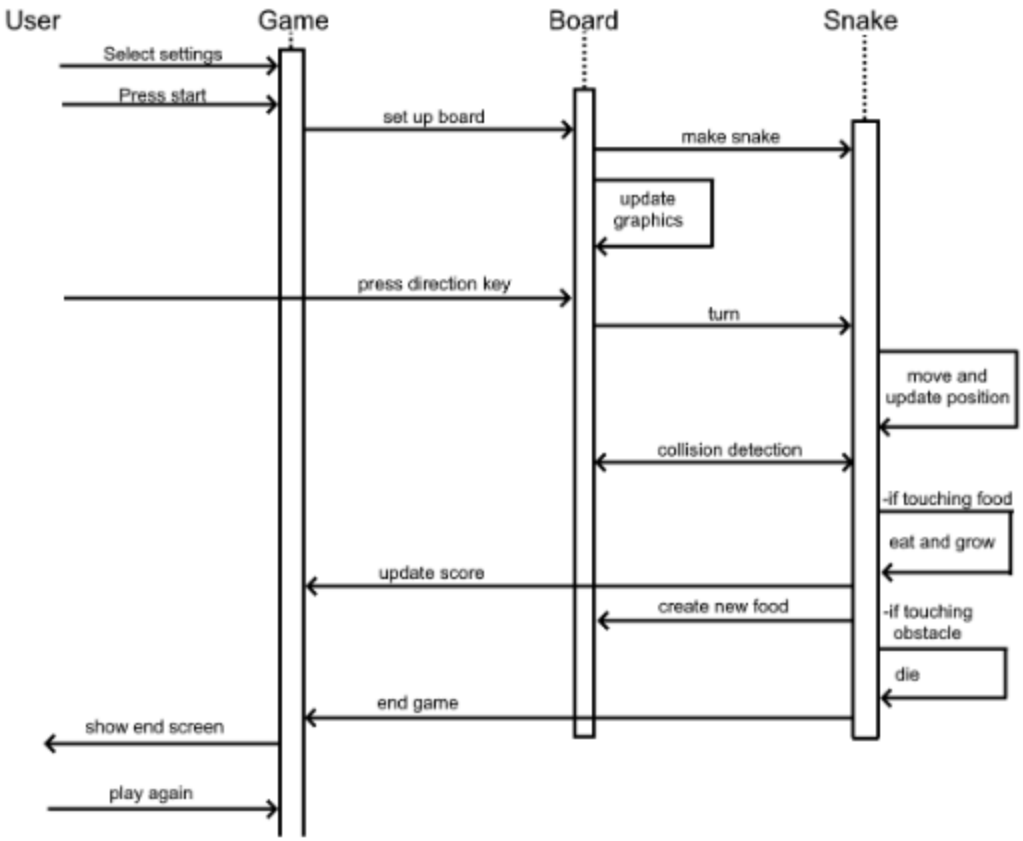
Bradley Russell is a Computer Science major who is currently a sophomore. He has experience with Java, C++, Python, JavaScript, and C#, and has made simple games using those last two. He also has an interest in dissecting the code of old video games to see how they work. On this project, he focused on the backend, making much of the snake and board classes, and made the game print to the terminal.

III. Project Requirements

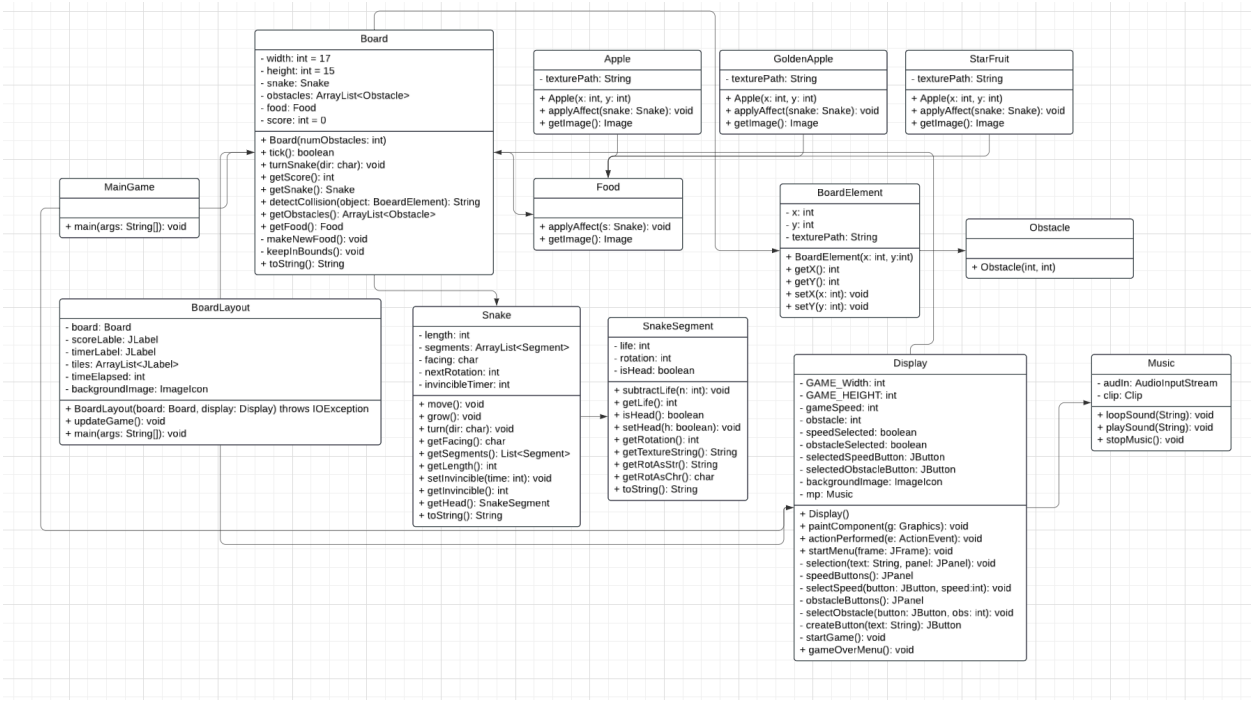
This section includes the major features and requirements we included for the project. The diagram below provides a clear overview of the game's functionality and design. These things were necessary in creating the game. The project requirements are organized in a diagram for readability.

Feature	Description
Snake Movement	Make sure the snake moves across the board smoothly in all directions using WASD and arrow keys
Snake Growth	Have the snake grow when it eats food
Food	An item that causes the snake to grow. - Apple grows by 1 segment - Golden apple grows by 5 segments - Star fruit grants invulnerability for 12 ticks
Collision	Detect when the snake collides with itself, a wall, or an obstacle on the board. If there is a collision trigger the ‘game over’ screen
Score	Display accurate player scores throughout the game. The score is the amount of food eaten.
Timer	Counts play time and correlates with the speed
Welcome messages	Display a welcome message when the game starts. Prompt the user to select a speed and obstacle difficulty
Difficulty Speed	Three levels of difficulty: Easy, medium, and hard. Easy: 200 speed Medium: 150 speed Hard: 75 speed
Difficulty Obstacles	Three levels of difficulty: Easy, medium, and hard. Easy: 0 obstacles Medium: 4 obstacles Hard: 8 obstacles
Game board	The area where the snake moves around with walls/barriers that can cause collisions.
Music	Plays music when the game starts
Game over message	Display the ‘Game over’ message Ask if the player wishes to try again, go to the main menu, and exit

IV. Solution Approach

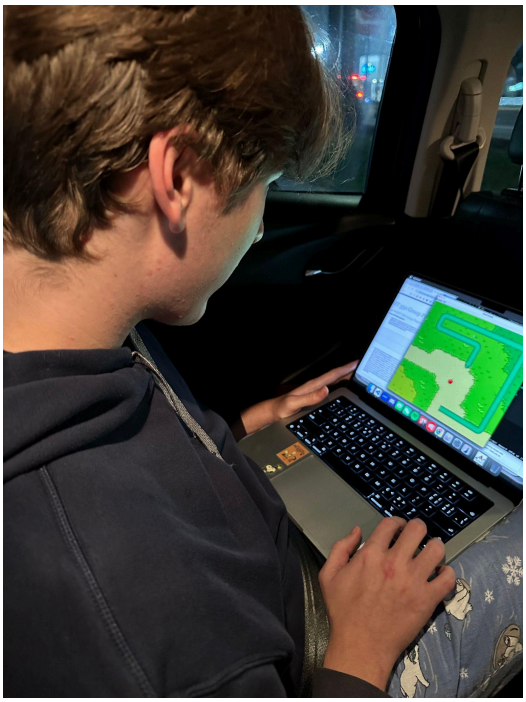


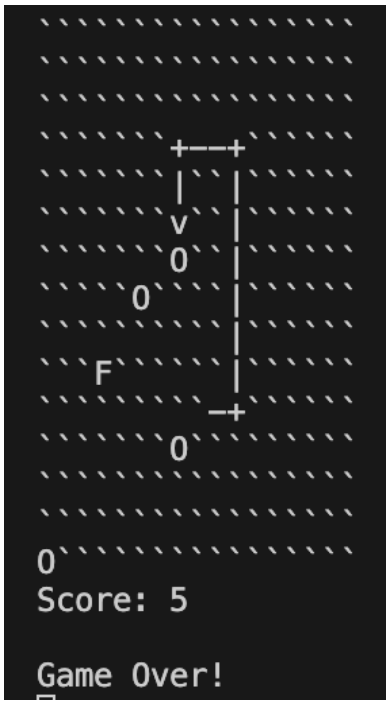
This is our simplified sequence diagram, it shows the core flow of interactions within the Snake game, demonstrating how the components (User, Game, Board, and Snake) work together to deliver the gameplay experience. The sequence begins with the user selecting settings and pressing "Start," prompting the game to initialize by setting up the board and creating the snake. During gameplay, the user presses direction keys (e.g., WASD or arrow keys), which the game sends to the board and snake to update the snake's movement and direction. The board continuously checks for collisions, determining whether the snake has eaten food, collided with an obstacle, or run into itself. If food is consumed, the snake grows, the score updates, and new food is generated. If a collision occurs with an obstacle or itself, the game ends and displays the end screen. Throughout the game, the board updates graphics to reflect changes in the game state, ensuring smooth gameplay. At the end, the user can choose to play again, restarting the sequence. This diagram effectively highlights the interaction between subsystems and key features such as dynamic snake movement, collision detection, and food mechanics, showcasing a modular architecture with clear responsibilities for each component.



Our UML diagram provides an overview of the class architecture for our Snake game, illustrating the relationships and responsibilities of various components. The Board class serves as the central hub, managing the Snake, Food, and obstacles, while handling collision detection and rendering the game state. The Food class is abstract, with subclasses Apple, GoldenApple, and StarFruit, each implementing specific behavior, such as granting invincibility or increasing the snake's length. The Snake class tracks the snake's position, movement, growth, and invincibility through its segments, represented by the SnakeSegment class. The Display class manages the graphical user interface, including the game board and menus, while the MainGame class initializes and controls the game loop. The BoardElement class acts as a base for objects on the board, like food and obstacles.

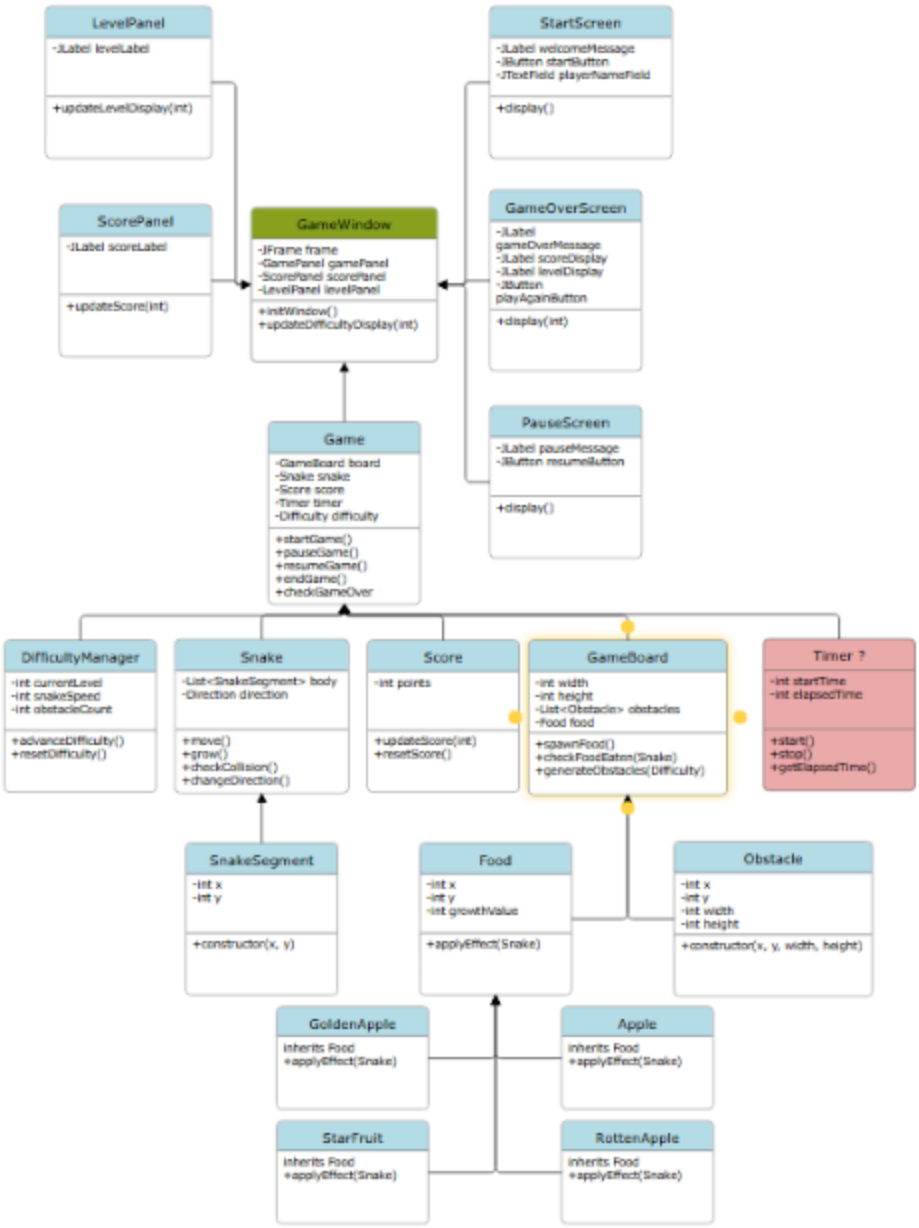
V. Test Plan





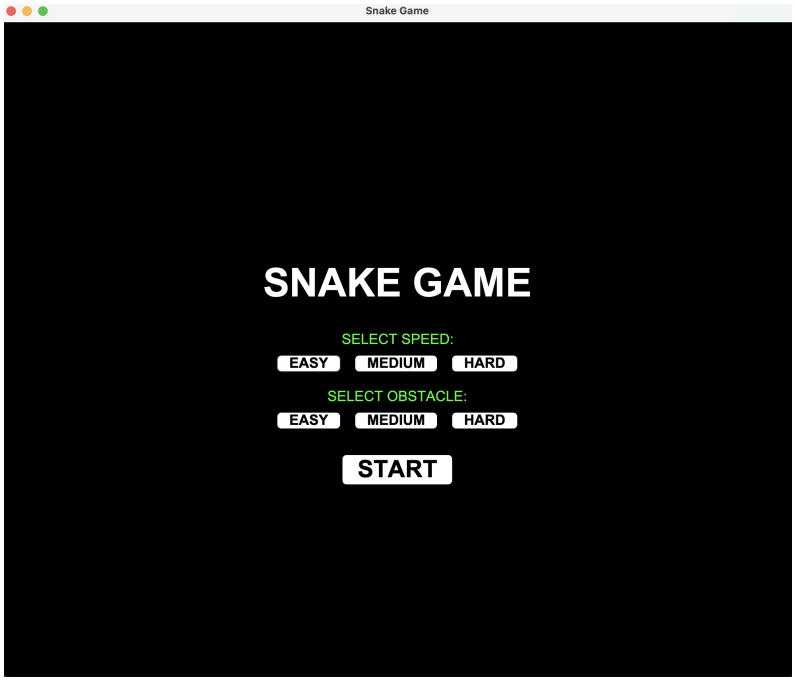
We had a friend test our project and he said that he really liked the design/visuals of the game but that sometimes the game lagged. Specifically, when the player tries to turn too fast, for example, if the user presses the up and right arrow too fast the snake will crash into itself. As for other tests, we did not have many. The only real test we had is that before we implemented GUI we had the game displayed in the terminal to make sure that we had a working game.

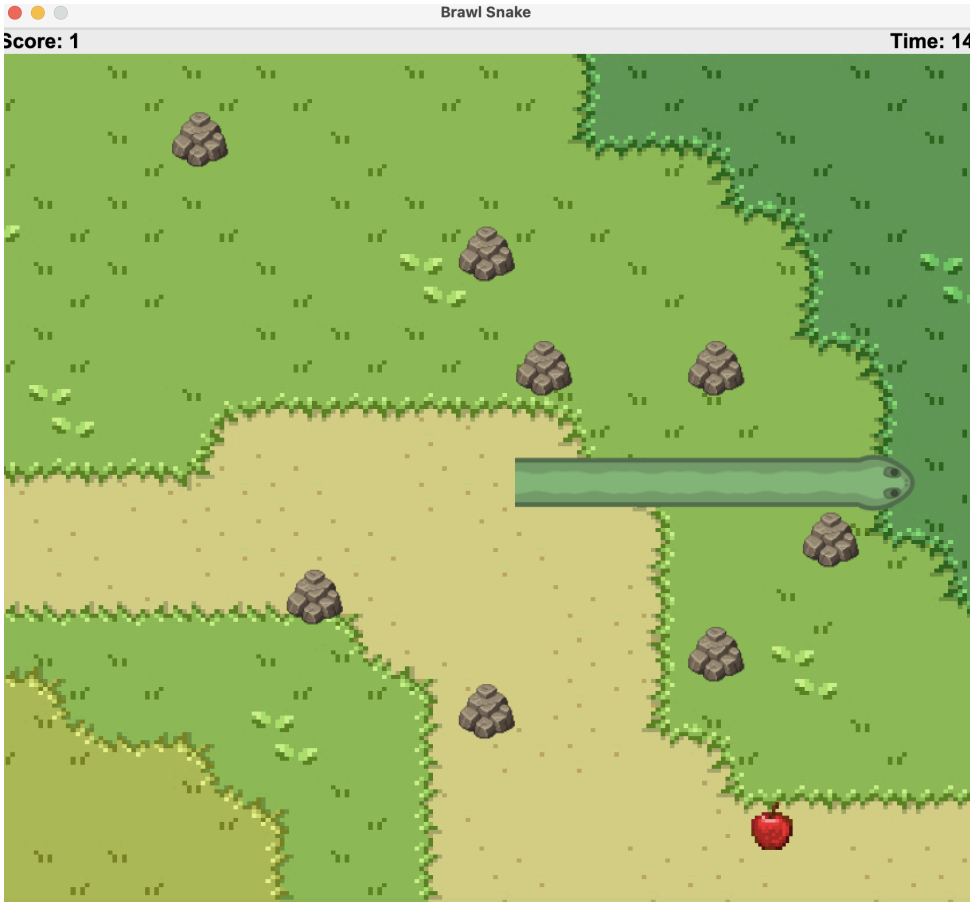
VI. Project Implementation Description



This was our original UML class diagram. We implemented almost everything that was planned barring a few exceptions. We wanted to add a timer because it seemed like a cool feature when Dr. Crandall was talking about it in class. While we did add it, it ended up not being a useful feature for the game. It just sits in the top right corner and counts how long the player has been on the level. We did a lot of the backend programming to start because we wanted the game to print to the console before implementing a GUI, during this we realized how messy and complicated a story mode and difficulty selection mode would be to implement together. We decided to go the more classic route of the original Snake game and only implement the difficulty level and obstacle level in the start screen. Another thing we tried implementing was the RottenApple subclass of the Food Class. It was originally going to remove a segment from the Snake if eaten, but we came to the decision it didn't fit the game. The whole point is to eat and grow and if there's no edible fruit on the board, only the rotten fruit and obstacles, it puts the

player in a spot to only lose, which we didn't want. With this, the difficulty levels are challenged by themselves, so the rotten fruit was not necessary.





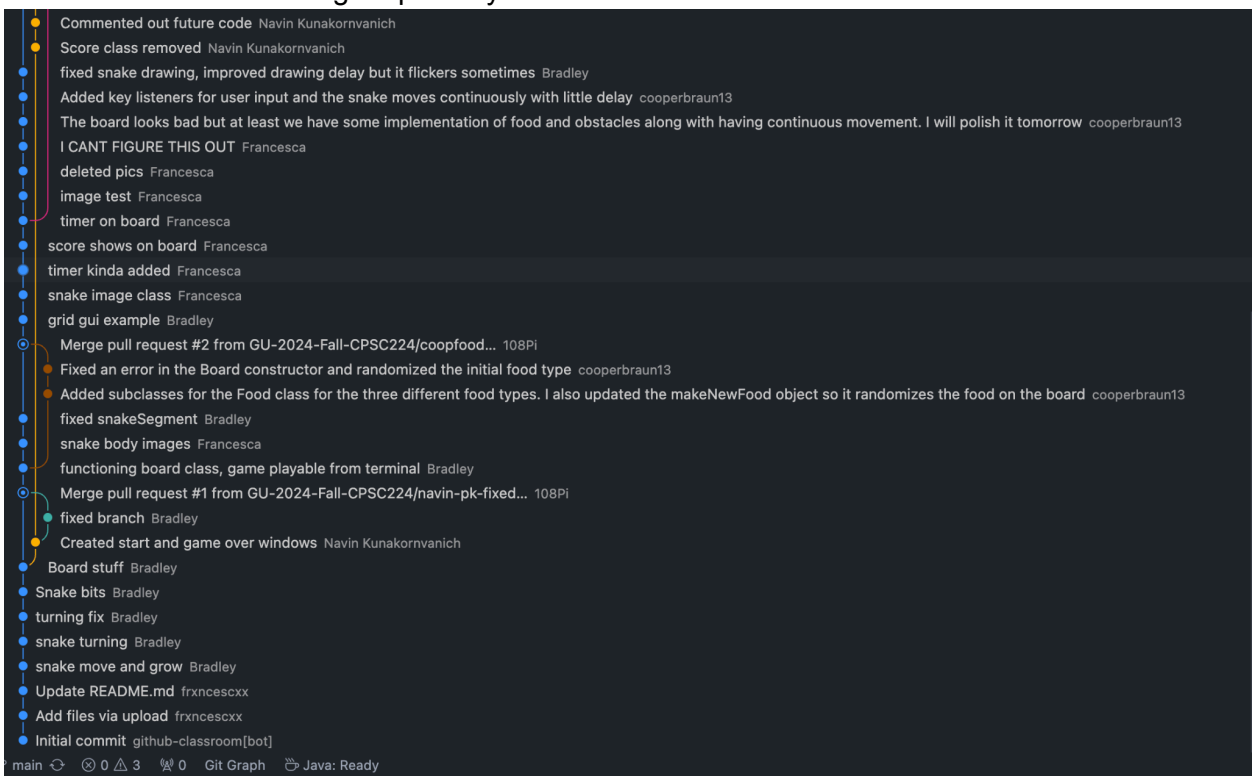
As you can see from the screenshots, we did end up finishing most of what we planned for. We ended up skipping a pause button. We still want to add this but sadly we ran out of time and it wasn't super high on our priority list. The idea of a pop-up screen for the pause button was also a little difficult with our time constraints. We started too late into our project so we had to scrap it. We also had music in our game, which we didn't fully plan for, but we did float the idea around. We wanted to add different sounds for the snake eating, colliding, etc. but we were running low on time so we have a good little soundtrack playing for the user. Lastly, I think a custom-made background to fit our game better would've been a really fun and intriguing concept, but again, it just wasn't a necessity for us.

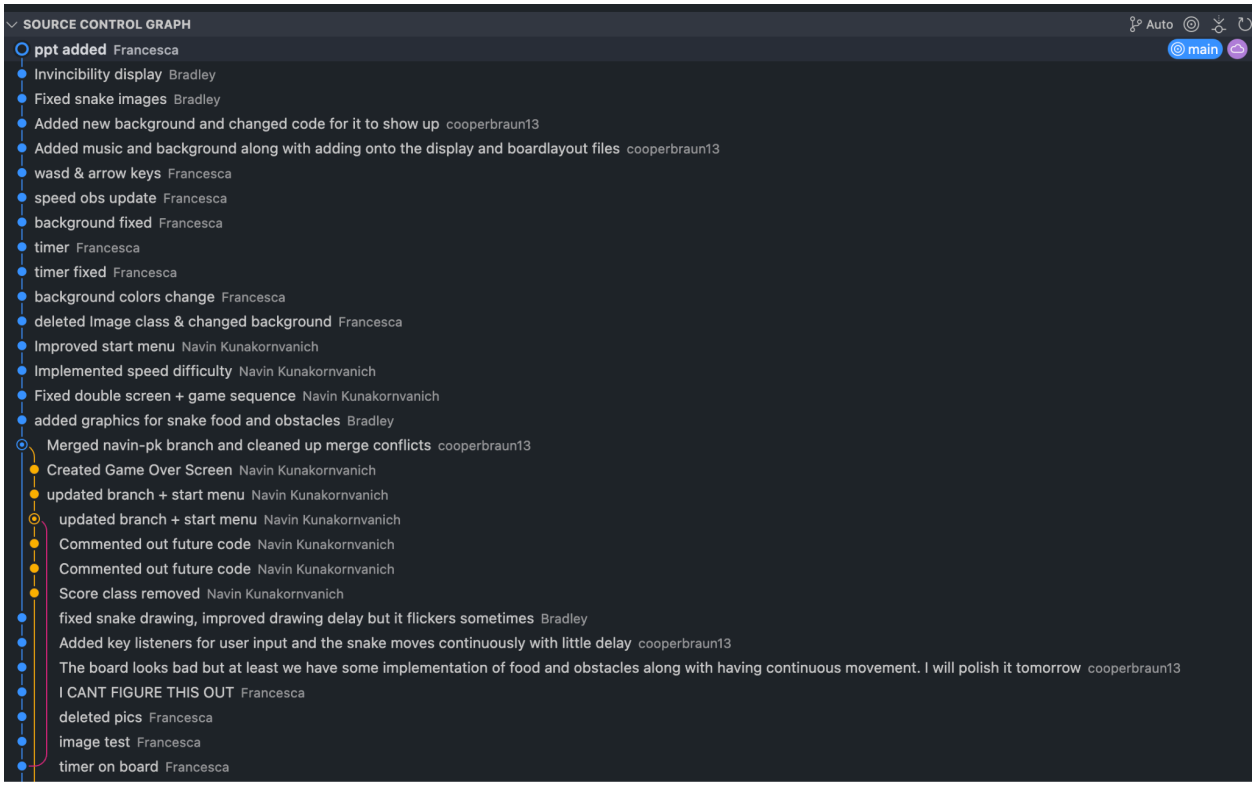
VI(a). Project Statistics

1) Number of total lines of code written by your team for the project
    a) 856 total lines in the source code
2) Estimated total lines of code written by **each** team member - this could include any lines deleted due to changes as you coded. Sometimes work is added and deleted as the project moves forward.
    a) Cooper - 400 (+300 -100)
    b) Fran - 850 (+500 -350)
    c) Navin - 650 (+400 -250)
    d) Bradley - 800 (+550 -250)
3) Amount of software tests like unit tests you wrote and/or had running.

    a) 1, only the console test
4) Number of commits each member generated
    a) 50 commits total
    b) Cooper - 7
    c) Fran - 18
    d) Navin - 10
    e) Bradley - 15
5) Number of branches used
    a) Cooper - 3
    b) Fran - 0
    c) Bradley - 1
    d) Navin - 1

Screenshot of our team's git repository commit tree





https://github.com/GU-2024-Fall-CPSC224/final-game-project-binary-brawlers

VII. Team Collaboration and Tools Used

To communicate we used email to share documents and created a group chat to text. As for meetings we did not have set meeting times, we would meet when we needed to. Overall, we met three times and did a bit of coding together but mainly individually. During those meetings, we reviewed each other's code and helped one another to fix the issues. Sometimes the code one person had would not work on their laptop but work on another's, same with commit. Some members used Git branches and pull requests and some did not. Since coding in a group is new to all of us, getting used to each other's coding styles was a new experience and presented us with some problems. Communication is needed to ensure that we do not overlap each other when coding. We did not use any planning tools for this project and we only had a couple of minor merge conflicts, nothing huge that caused a setback.

VIII. Future Work

Right now our ‘Main Menu’ button is buggy so ideally we would fix that. As for features, we would add if we had more time, we would change all the graphics (snake, food, background, obstacles) based on the difficulty that the user selects. We would also implement difficulty differently to allow for the graphics changes. The difficulty is currently separated by speed and obstacle count but we would change it to just ‘easy’, ‘medium’, and ‘hard’ and set the speed and obstacle count ourselves.

We would also want to add a story mode, or an iterative process through the game with different levels that had different graphics. Overall, fixing some of the bugs would be the most important part but I think we have something to build off of and improve.

IX. Glossary

UML (Unified Modeling Language): A standardized visual modeling language used to specify, visualize, and document the architecture and design of software systems.

Git: A distributed version control system used for tracking changes in code and coordinating work among multiple developers.

Repository (Repo): A storage location for your project files and their version history, tracked by Git.

Branch: A parallel version of a repository, allowing developers to work on features or fixes independently of the main codebase.

Push: The process of uploading local commits to a remote repository.

Pull: The process of downloading changes from a remote repository and integrating them into the local branch.

Fetch: The process of downloading changes from a remote repository without integrating them into the local branch.

Merge: The act of combining changes from different branches into one.

Rebase: A method of integrating changes from one branch into another by replaying commits on top of the target branch.

Clone: The act of copying an entire remote repository to your local machine.

Fork: A personal copy of a repository, typically used to propose changes to someone else's project.

Pull Request (PR): A GitHub feature used to notify collaborators of changes in a branch and request code review before merging.

Commit: A snapshot of changes made to files in a Git repository.

Branch: A parallel version of a repository used in Git for independent development without affecting the main codebase.

Pull Request: A GitHub feature allowing developers to notify team members of changes in a branch and request a review before merging them into the main branch.

Java Swing: A Java-based GUI toolkit used to create desktop applications.

GUI (Graphical User Interface): A user interface that allows interaction with electronic devices using graphical elements like windows, buttons, and icons.

Star Fruit: In the context of the game, a type of food item granting invincibility for a limited duration (10 ticks).

Tick: A unit of time or iteration in a game's loop, typically corresponding to one update of the game state.

Console Test: Testing the program through the command-line interface before implementing a graphical interface.

Backend: The part of the application managing the logic and data processing, is not directly visible to the user.

Frontend: The user interface part of the application that interacts with the user.

Game Board: The area where gameplay occurs, containing the snake, obstacles, and food items.

Merge Conflict: An issue arising in Git, when changes from different branches, overlap and cannot be automatically merged.

X. References

GeeksforGeeks. "Java KeyListener in AWT." *GeeksforGeeks*. Last modified March 10, 2021. https://www.geeksforgeeks.org/java-keylistener-in-awt/

Oracle. "Creating a GUI with Swing." *The Java Tutorials*. Accessed December 12, 2024. https://docs.oracle.com/javase/tutorial/uiswing/

GitHub. "Using Git." *GitHub Documentation*. Accessed December 12, 2024. https://docs.github.com/en/get-started/using-git.