COMP 055-02-Application Development

SUPER MARIO LITE

Report 3: Final Report

by

Murad Haider, Ade Sampson, Kunal Babbar, Kaung Khant Pyae Sone, Michelle Vu

December 9, 2020

GitHub: https://github.com/ClassCOMP55/groupproject-team-5-devil-s-advocate

Basecamp: https://3.basecamp.com/4838143/projects/18904364

Table of Contents

Individual Breakdowns

	Michelle Vu	Kunal Babbar	Murad Haider	Ade Sampson	Kaung Khant Pyae Sone
Cover Page – 2pts	1		0.5	0.5	
Individual Break – 3pts	1	1			1
Table of contents – 1pt			0.5	0.5	
Key Revisions – 6pts	1	1.5	1	1	1.5
Customer Statement – 1pt			1		
Glossary – 1pt					1
Functional Reqs – 3pts	0.5		0.5	1	1
Interaction Diagrams – 8pts	2	2	2	2	
Class Diagrams – 9pts	3	0.5	1.5	1	3
System Architecture – 1pt				1	
Algorithms/DataStructs – 1pt					1
UI Design – 3pts		1.5	1.5		
History of Work – 4pts	1	1	1	1	
Conclusions – 5pts	0.5	1.5		2	1
References – 3pts		1	0.5		1.5
Total pts each member	**10**	**10**	**10**	**10**	**11**

Summary of Changes

Provide an **itemized list of key revisions** since the previous two reports. This includes key changes in *project objectives*, *use case descriptions*, and *system design* (e.g., interaction diagrams, class and use case diagrams).

Project Objectives:

The COVID theme was removed from implementation because of the level of difficulty associated with the beginning of the project. The group's inexperience with JAVA and its lack of familiarity with a software design cycle played a key role in this decision.

Use Case Descriptions:

As for the changes of the use case descriptions, we removed some features such as the total live count, timer, point count, coins, and turtles, which were all mentioned in our old use case of the actual game play. Furthermore, without the implementation of the coins and point system, we found it unnecessary to include a user login to the beginning of the game to track the highest scores. Due to the timeline of the project, we had to simplify the number of features to now consist of multiple goombas and a long level instead of implementing multiple small levels.

System Design:

We initially planned to create levels by hardcoding them into a class file filled with arrays of Entities, Sprites, GImages, GRects and other things that are necessary for the game to work. However, we found out about a much better solution: using TMX files to load the levels into our game and sprite sheets to store the images. With the help of the TiledReader library, it allowed us to use the Tiled editor to create levels visually and use the sprite sheets to store most of the assets of a single image file.

Class Interaction:

The original formatting proposed for the Entity class was redundant and inefficient. It required the use of classes that could have and were turned into array lists. This allowed for code that is faster, more readable, and less prone to errors. This change is documented, and can be compared to the original idea on the UML diagrams.

Customer Statement of Requirements

We know how you’re feeling during this year of COVID-19. You must endure the slow creep of boredom as time passes by, the lack of motivation to get tasks accomplished, and the draining responsibilities of work or school.

These conditions seem to be determined to join forces, with the sole goal of raising your stress levels to all-time highs. We wanted to immediately tackle these feelings, and bring some simplicity and enjoyment back into your lives! Everyone loves the feeling and comfort of familiarity, and the excitement of something new and refreshing. That is why we wanted to revive the classic game of Mario by incorporating brand new visuals *and obstacles*!

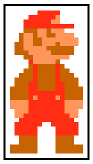
Mario has been a dominant player in the global gaming community since the late 1980s. This level of notoriety allows us the chance to build a program that is firmly entrenched in the zeitgeist of the information age.

Located below is an example screenshot of the game. The original Mario game was written in an assembly language for the 6502 microprocessor. Thankfully we won’t have to go that route. We can take you, the user, on Mario’s latest adventure as we look to recreate the program in the JAVA programming language.

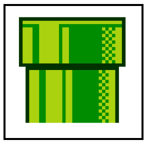
Glossary of Terms

TMX: Translation Memory Exchange is a file format standard.

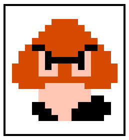
Mario: Mario is depicted as a portly plumber who lives in the fictional land of the Mushroom Kingdom. He is our main character, the character that our player will control to progress through the game.



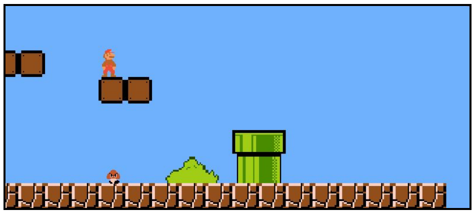
Pipe: A type of obstacle which resembles a green pipe that restricts the movement of our character as it comes up occasionally in the game.



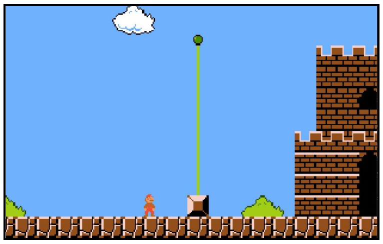
Goombas: They resemble small, brown mushrooms and are a fungus-based species. Goombas are physically weak and are not much of a threat to the player since a single stomp usually defeats them.



Platform: In the game, there are platforms that our character can jump on, to reach higher places, or to traverse areas where there are too many obstacles/enemies and it is wiser to avoid them.



Flag And Castle: This is the goal situated at the end of the level. The flag is a flagpole where there player has to jump onto the flagpole to win the game



Functional Requirements Specification

a. Casual Description

The use case will explain every step the user needs to take to complete the game, what the user would need to do if there's an enemy or any type of obstacle in the way. It basically shows a step-by-step version of how the actual game would work.

b. Main Use Case

This Use Case tells how the program works, when to decide the win and how to start the game. It basically overviews all the other use cases.

1. Screen of welcome to Mario with new game option and high score on the top
2. User Clicks on the New Game
3. The Mario game screen comes up with a message saying welcome and the controls/instruction for the game.
4. Presses any button to start.
5. The game screen comes up with the Mario player on it.
6. Player moves the Mario
    a. Player moves Mario to the left
        i. Mario does not move, he stays on the starting position
        ii. Player Jumps Mario again while in the air
            1. Mario does not jump any higher i.e it won't double jump
        iii. Player moves to the left and jumps
            1. Mario just jumps and lands straight
    b. Player Jumps Mario
        i. Mario jumps and lands on the same position.
        ii. Player jumps and moves to the left while in the air
            1. Mario jumps and lands straight
        iii. Player jumps and moves to the right while in the air
            1. Mario jumps to the right
    c. Player moves Mario to the right
        i. Mario meets the barrel
        ii. Mario meets an enemy
        iii. Mario encounters a platform
7. Mario encounters the flag
    a. Player jumps the Mario on the flag
        i. Gain points according to the height where he jumped on the flag
        ii. Animation showed how mario is is going inside the castle
8. Game Over
    a. Game closes

c. Main Use Case - Obstacles

This use case is for a particular type of case which is when the player encounters an obstacle, like a barrel. Basically it tells about all the cases of how the user can interact with a barrel and how the player needs to tackle the barrel.

1. Mario won't be able to move any further to be able to move Mario further the player needs to jump over the barrel.
    a. Player moves Mario to the left
        i. Then again it need to move to the right to continue the game
    b. Player jumps Mario
        i. Mario jumps up and comes back to the same position
    c. Player moves Mario to the right
        i. Mario jumps up and comes back to the same position
    d. Player moves Mario to right
        i. Mario does not move.
    e. Player jumps Mario to the right
        i. Mario jumps on the barrel. Barrel is just an obstacle to make the Mario jump in order to cross it
            1. Player moves Mario
                a. To the left
                    i. Falls to the left of the barrel
                b. Jumps Mario
                    i. Just jumps on the barrel vertically
                c. Jumps left
                    i. Jumps left to the barrel and falls off the barrel
                d. Moves right
                    i. Falls off the barrel
                e. Jumps right
                    i. Mario jumps off the barrel to the right of it

d. Main Use Case - Enemies

When Mario is close to an enemy, he can fall on top of them to remove the enemy from the map or avoid colliding the enemy. If Mario collides with an enemy when Mario is not falling, then Mario will restart the level.

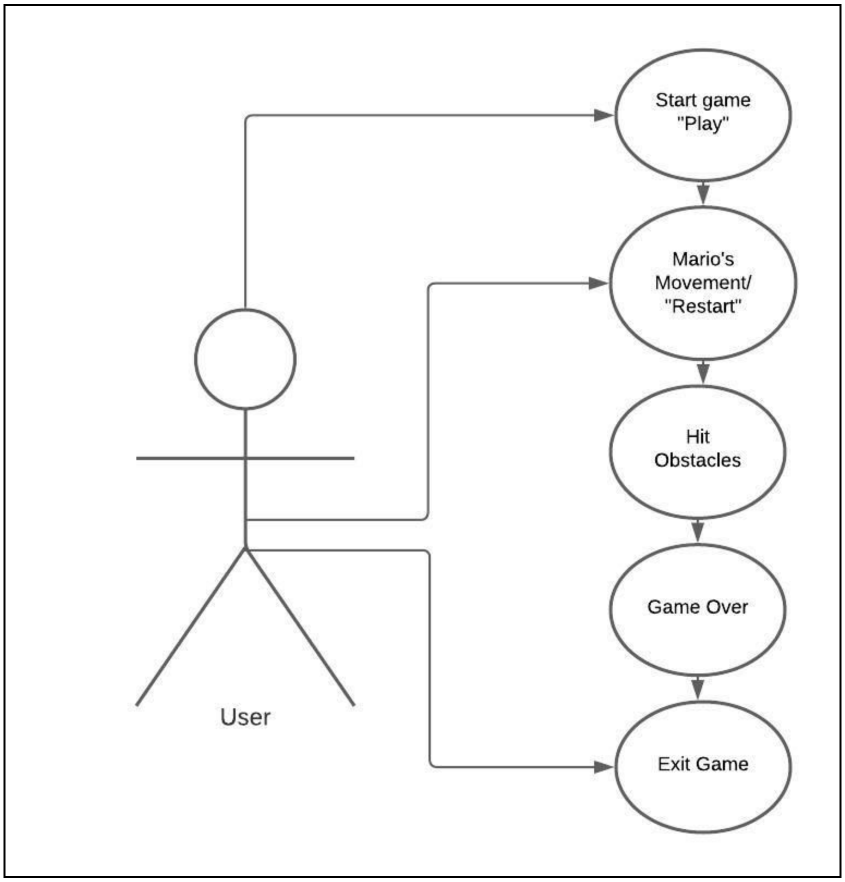
1. Player moves the mario towards the enemy
    a. Mario collides with the enemy
        i. Mario dies, and the death screen with an option to restart the game is shown.
2. Player jumps towards the enemy
    a. Player jumps on the enemy
        i. Mario "bounces" off of enemy vertically

        ii. The enemy is killed
        iii. An animation of the enemy smashed appears, then the enemy disappears
    b. Player jumps before the enemy
        i. Mario dies, and the death screen with an option to restart the game is shown.
    c. Player jump over the enemy
        i. Mario moves to the right and continues the game
        ii. Mario turns back and jumps

e. Main Use Case - User Interface

This is the use case for the user interface.

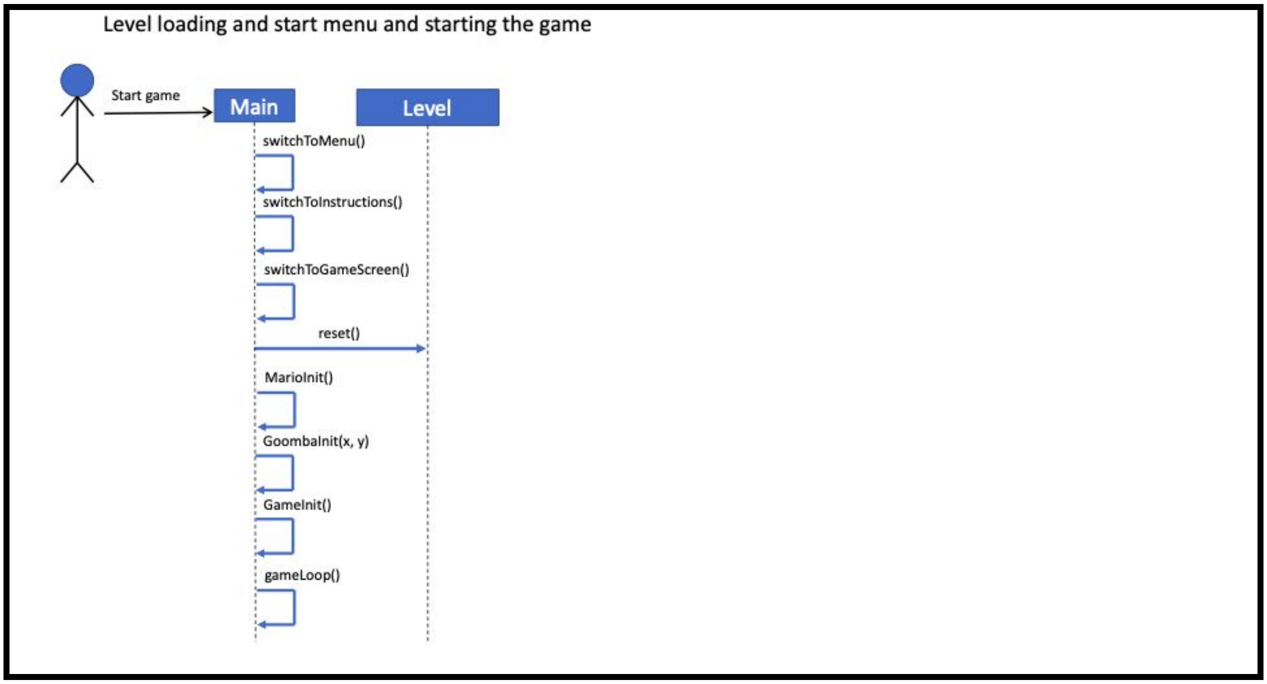
1. Menu:
    a. Start Game Screen/Button
2. Level:
    a. Level setup
    b. Win (back to menu or play again)
    c. Lose (back to menu or play again)

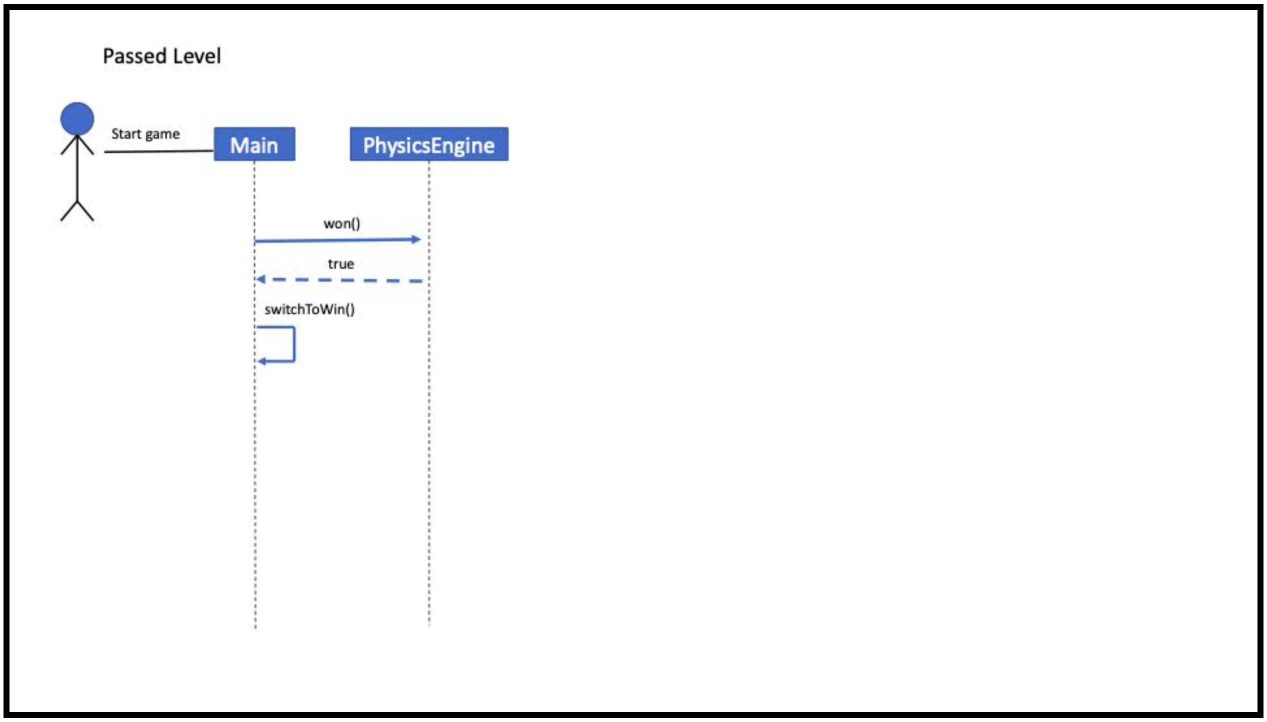


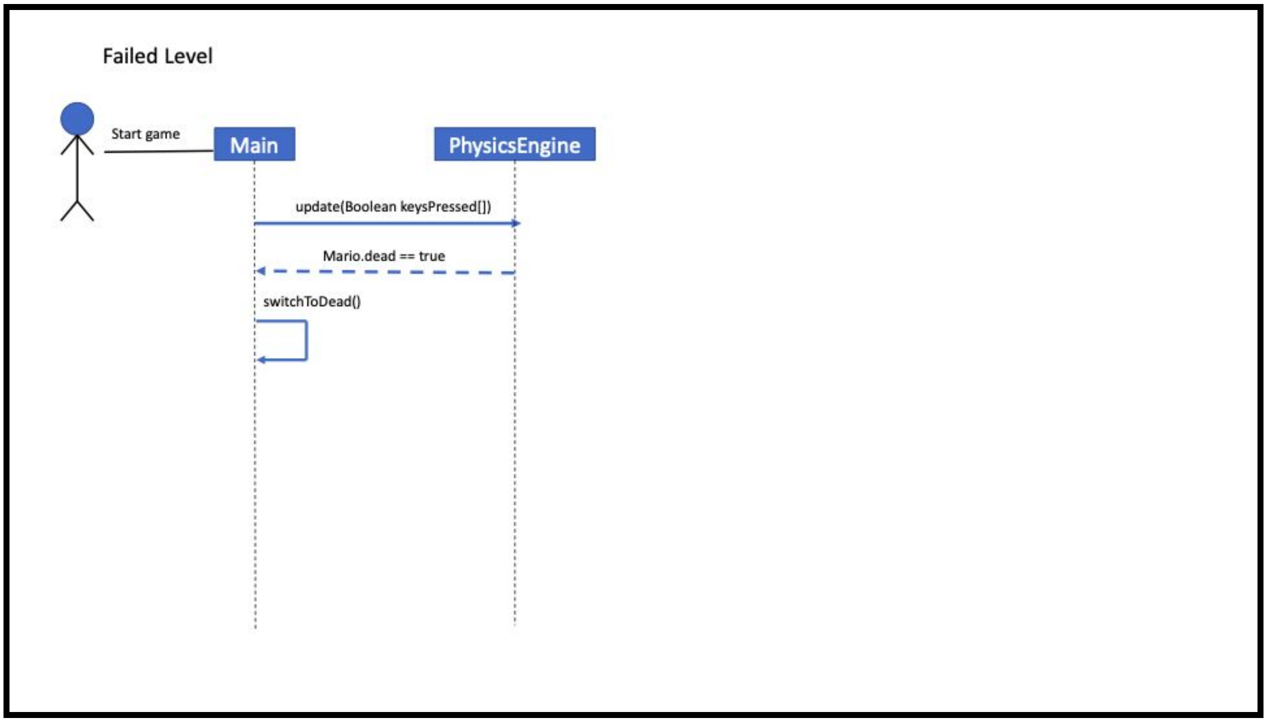
Interaction Diagrams

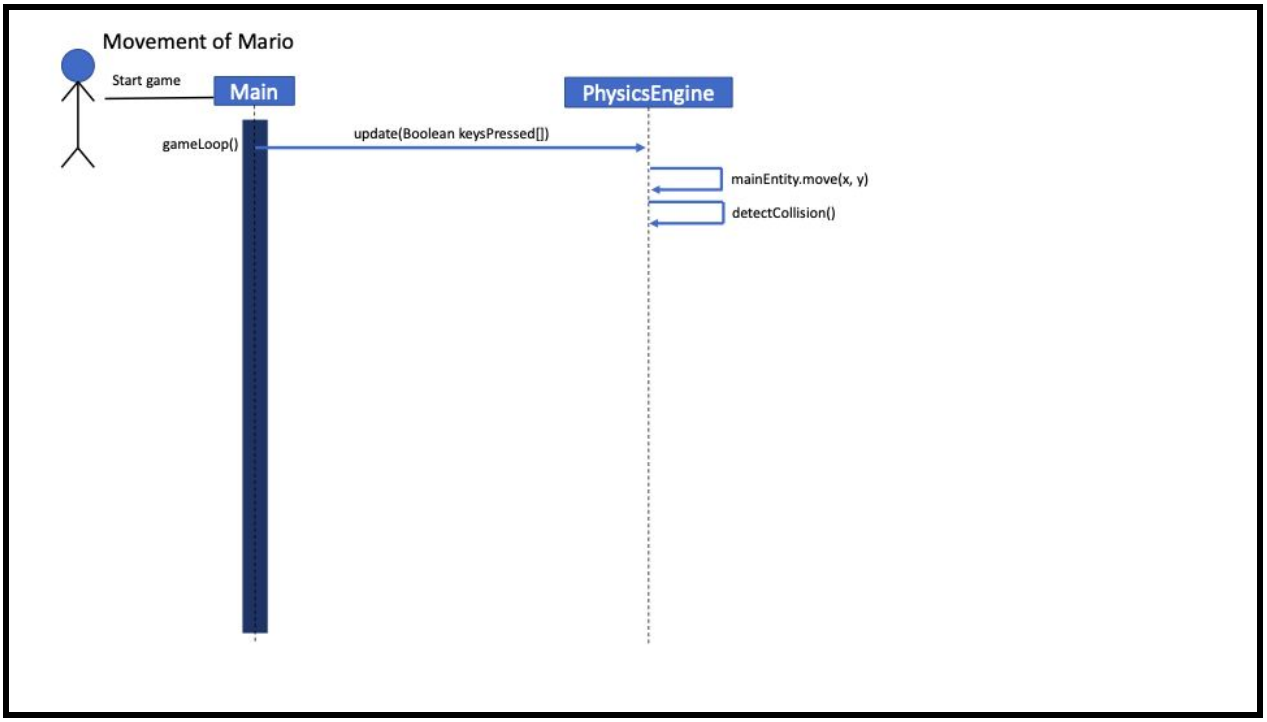
Only the first sequence diagram is changed. The implementation of the TMX files dictated the use of new methods to ensure compatibility. For the reasoning and justification, please refer to the Summary Of Changes on page 4.

Sequence Diagram:






Failed Level
Start game
Main
PhysicsEngine
update(Boolean keysPressed[])
Mario.dead == true
switchToDead()


Movement of Mario
Start game
Main
PhysicsEngine
gameLoop()
update(Boolean keysPressed[])
mainEntity.move(x, y)
detectCollision()

Class Diagram and Interface Specification

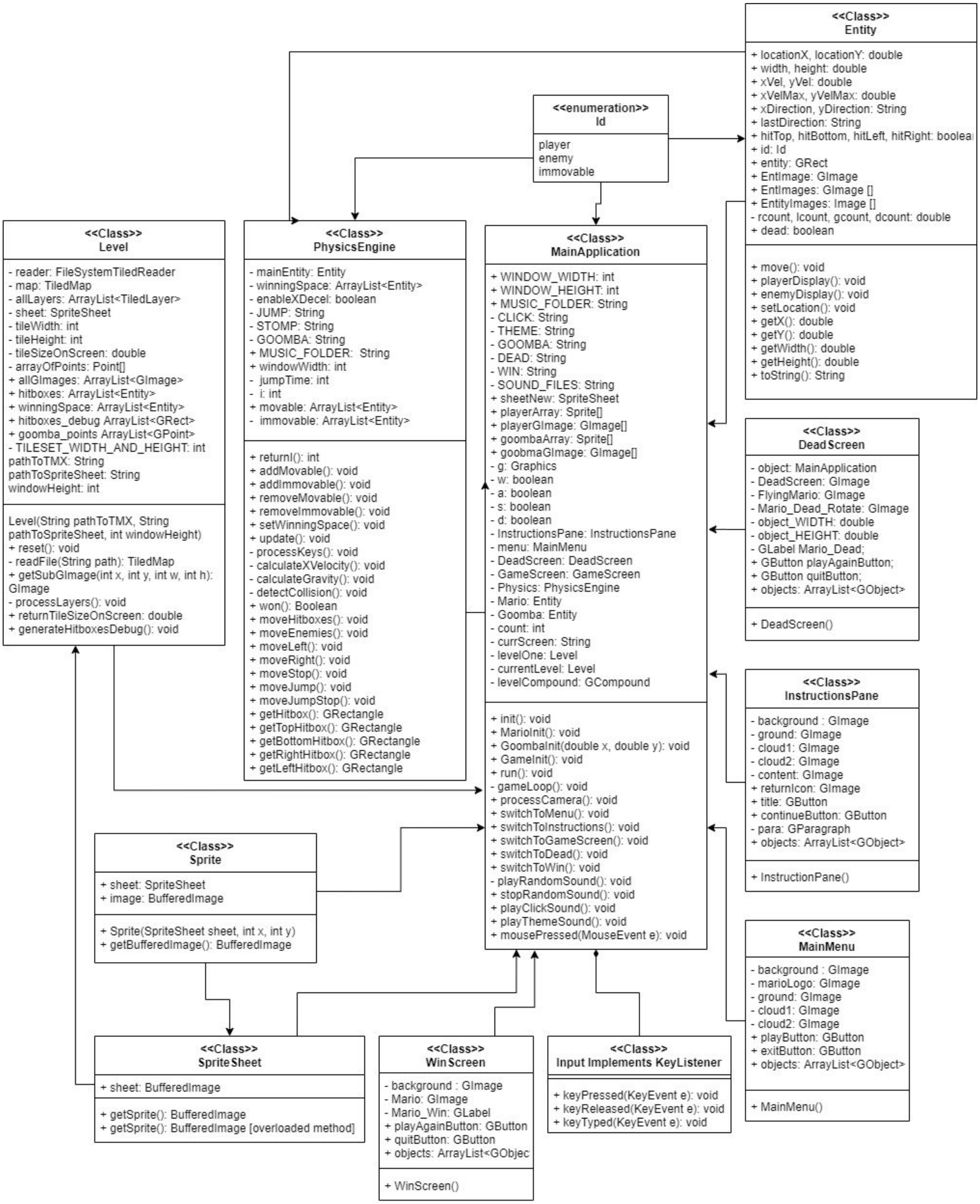
1. The game's PhysicsEngine class, is allocated data pulled in from the MainApplication class. It then is able to perform independently from any other class to perform its designed capability of ensuring Mario is able to move in a manner consistent with the original Mario game. MainApplication functions as the implementation of high cohesion, whereas PhysicsEngine directly proves the application of low coupling.
2. The DeadScreen class is called from MainApplication, and provides the user with images and text, which inform the user of their characters' untimely death. This class is independent from other classes, however, it does borrow protocol from the MainApplication object. This is another illustration of code compartmentalization, which is championed by the low coupling design principle.
3. The final example consists of the meshed functionality of the Sprite and SpriteSheet classes. This is an example of a higher level of coupling. The two classes work together to pull a single image from a .png file that holds a collection of images. To parse and select the correct image, these two classes work together. Sprite actively calls the getSprite() method from SpriteSheet, allowing both classes to work together seamlessly. While adhering to the high cohesion, low coupling principle, it remains important to write code that efficiently gets the job done. And retains elements that allow for easy maintenance and upgrades.

(UML Diagrams below)

New UML Class Diagram

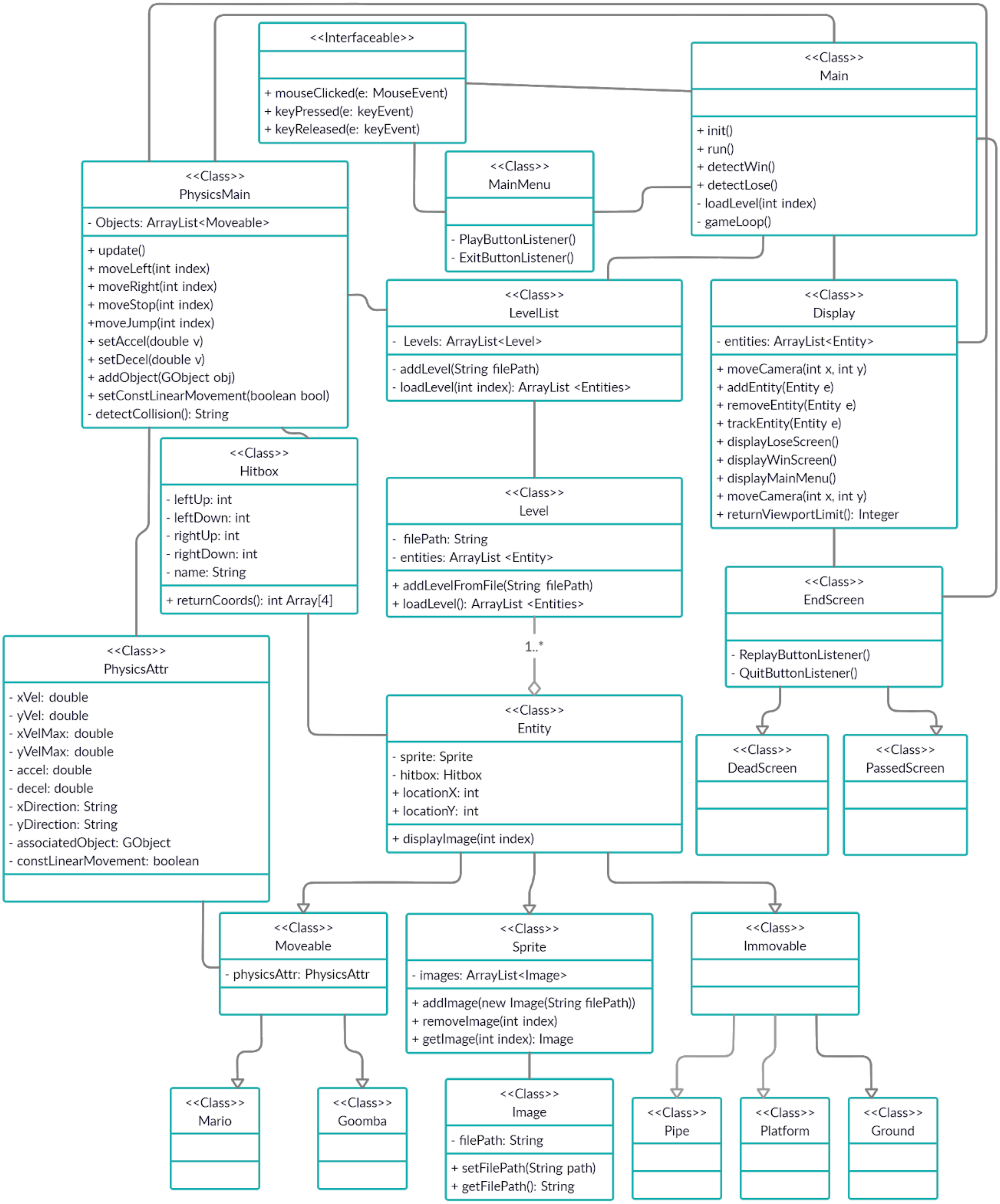
Class Diagram:

https://drive.google.com/file/d/1G089wUTvqxPxO_FqcyAkp-i8HhFq7djU/view?usp=sharing



Old UML Class Diagram

Class Diagram: https://app.creately.com/diagram/BOmnt9Biofh/view



System Architecture and System Design

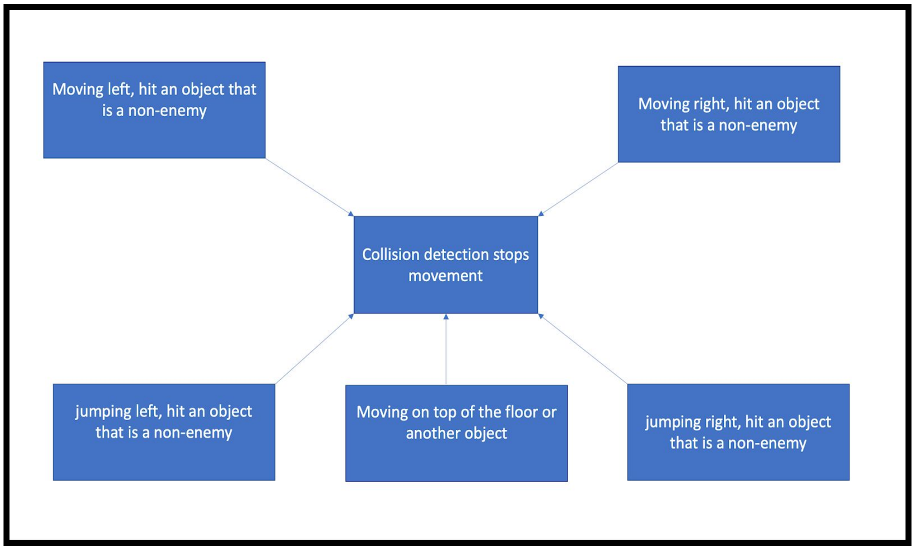
Hardware / Software Requirements:

- Display
  - Color
  - 800 x 600 pixels required
- Storage
  - At least 200 megabytes for this program and java dependencies
- A keyboard and mouse for input
- Any processor that supports JDK 8 and is supported by Windows XP and above
  - Pentium-2 266 MHz or higher
- At least 1 gigabyte of RAM
- Any graphics processor that is supported by Windows XP and above
- Operating System
  - Windows XP and above

Required subsystems mappings to hardware: none. This game is designed to be run on a single machine.

Algorithms and Data Structures

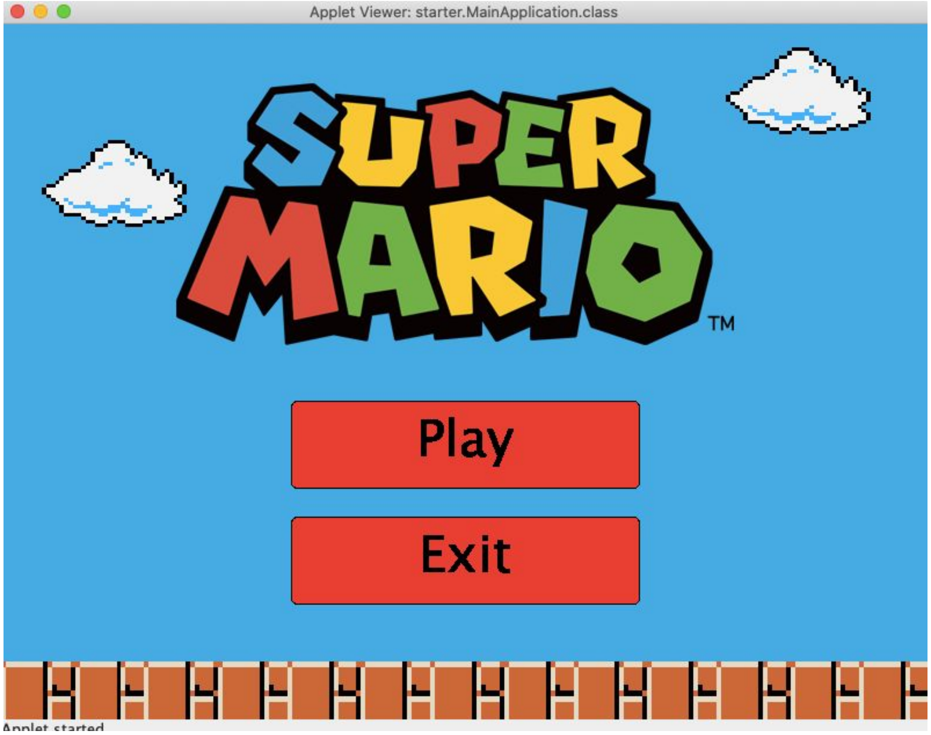
1. Algorithms
    a. The most complex algorithm deals with the program response to object collisions. The team is actively trying to develop a framework for this issue before coding officially begins. Below is an activity diagram of what the algorithm will be tasked with accomplishing.

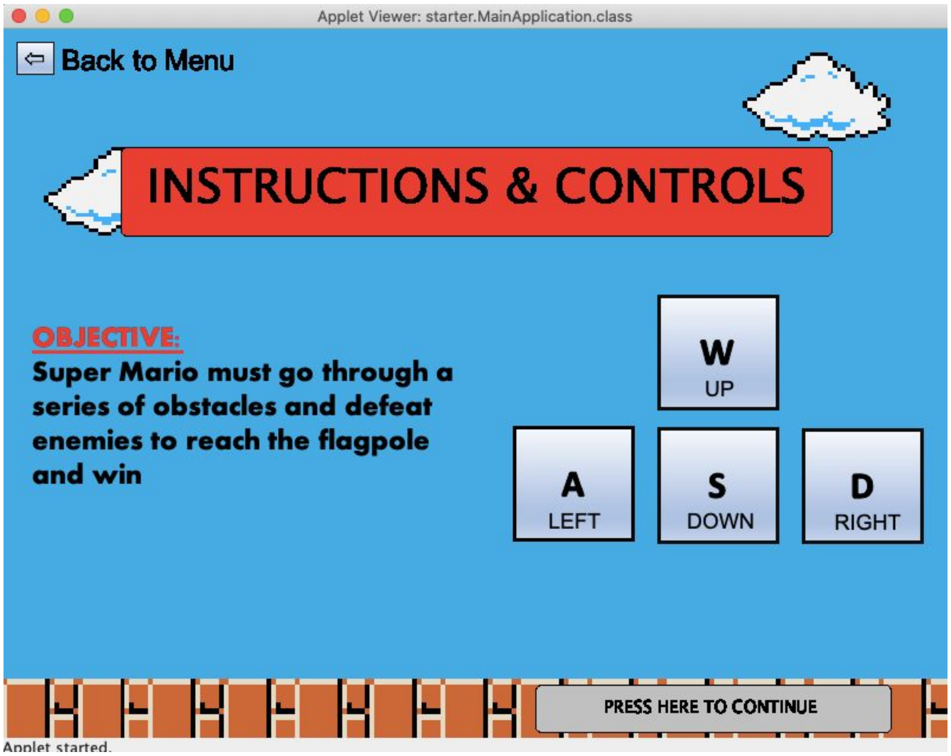

Moving left, hit an object that is a non-enemy
Moving right, hit an object that is a non-enemy
Collision detection stops movement
jumping left, hit an object that is a non-enemy
Moving on top of the floor or another object
jumping right, hit an object that is a non-enemy

2. Data Structures
    b. We will use arrays and array lists. These were chosen for performance and simplicity. These choices may evolve to more complex structures based on the ease of use and desired outcome. At this time, using a more complex data structure isn't justified based on the project's use-cases.

Array-List uses an array as its underlying implementation, so in effect you have the advantage of faster access to elements, while at the same time you can dynamically add and remove elements. We will be using Array-Lists to store the elements from the TMX file such as, GImages, hitboxes etc. We will use Arrays to store the images interpreted from the sprite sheets inside the entity class. This would make it much easier when the user wants to restart a level from the beginning, and that is the reason we use this data structure. We can just call that certain array for that point to restart the game and return all the objects into its original position. Array Lists allow for direct reference to every element in the list. Array lists are faster than linked lists which will make our program run more efficiently. We also used ArrayLists because we didn't have to deal with indexes while deleting or modifying existing objects as ArrayList contains a built-in search function for the functions we use.
One more advantage is that it is more flexible changing the underlying implementation of an ArrayList, for eg, if you want to make it synchronized, you can convert to a Vector without having to rewrite all the code for an Array.

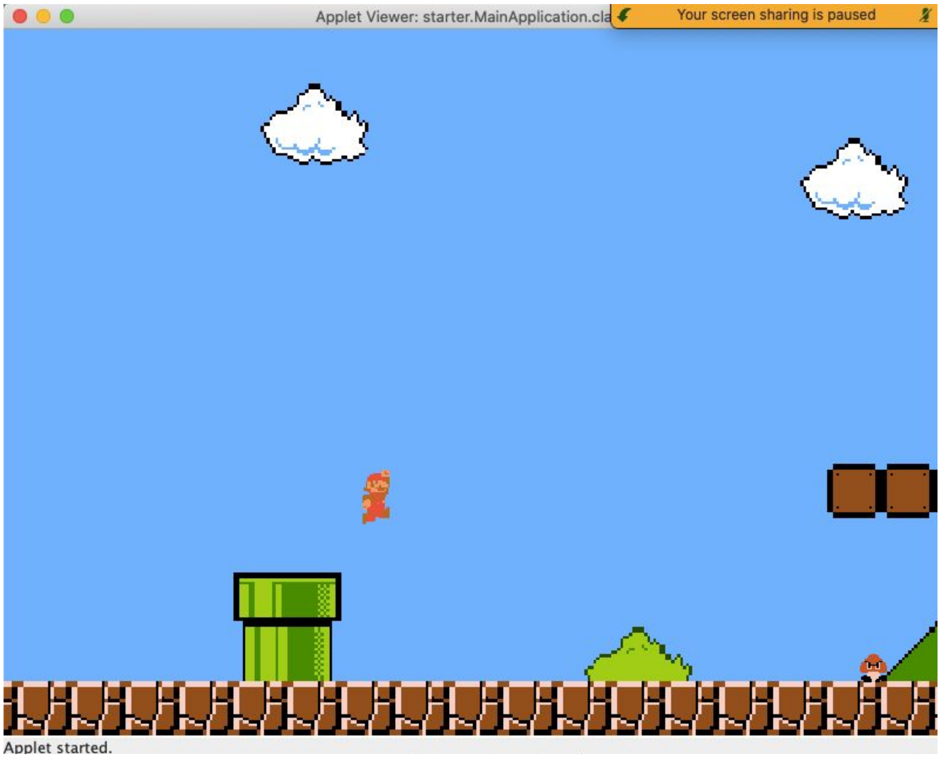
User Interface Design and Implementation



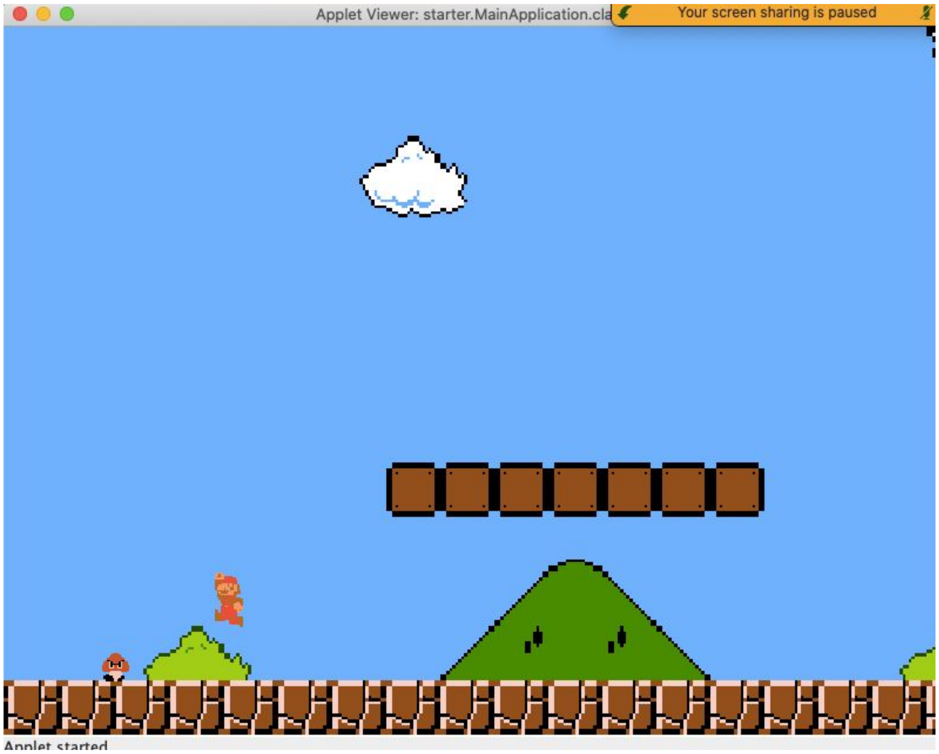


OBJECTIVE:

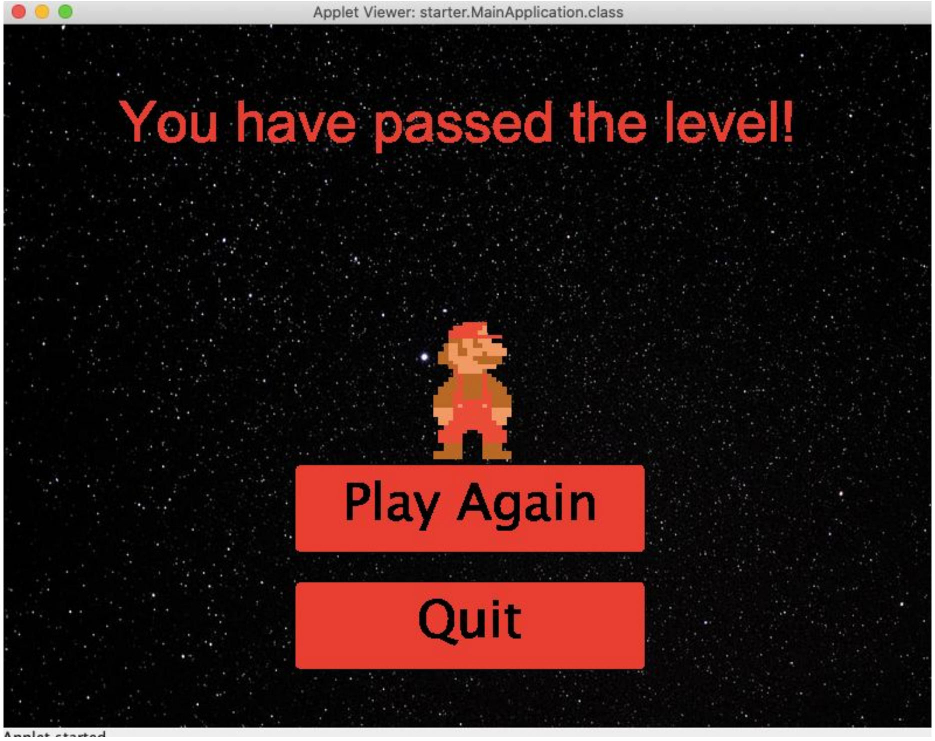
Super Mario must go through a series of obstacles and defeat enemies to reach the flagpole and win


Applet Viewer: starter.MainApplication.cla
Your screen sharing is paused

Applet started.

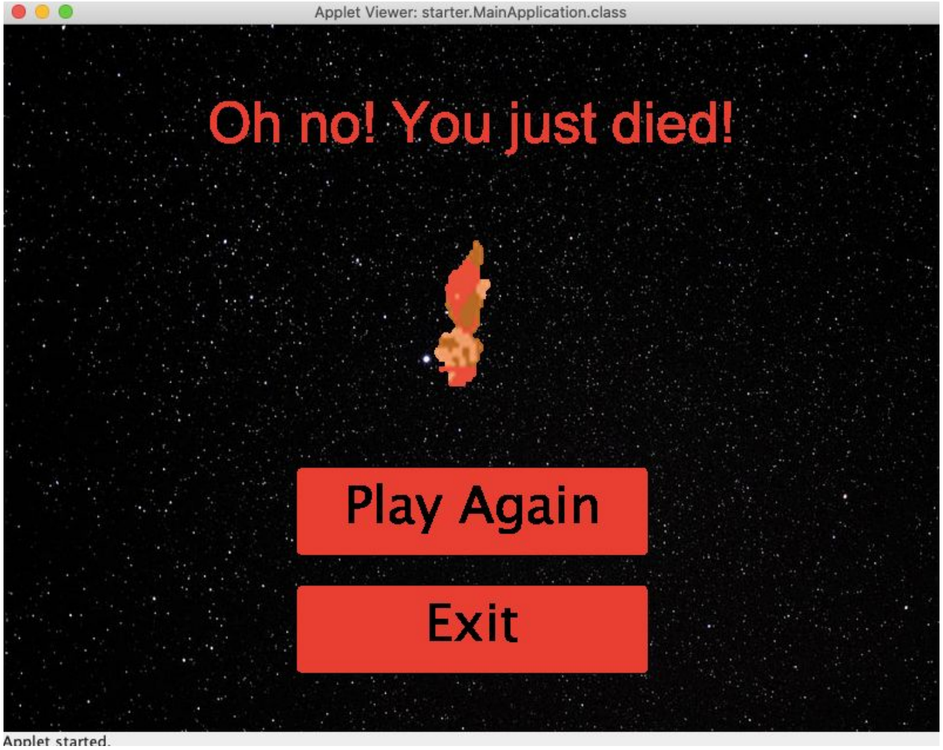

Applet Viewer: starter.MainApplication.cla
Your screen sharing is paused

Applet started



You have passed the level!

Applet started.



Oh no! You just died!

Applet started.

History of Work & Current Status of Implementation

History of Work:

Week:	Report #1	Report #2	**Report #3**
Week 1 (Sept 28-Oct 2)	☑ Project Proposal (Sept 27) ☑Team Charter (Sept 27) □ Requirements Analysis (Oct 4)	☑ Project Proposal (Sept 27) ☑Team Charter (Sept 27) □ Requirements Analysis (Oct 4)	☑ Project Proposal (Sept 27) ☑Team Charter (Sept 27) ☑ Requirements Analysis (Oct 4)
Week 2 (Oct 5 -Oct 9)	□Research Materials?	□Research Materials?	☑ Research Materials? ☑ Paper Prototype(Oct 11)
Week 3 (Oct 5 -Oct 9)	□ Start Project	□ Start Project	☑ System Design Report (Oct 18) ☑ GitHub Assignment
Week 4 (Oct 12 -Oct 16)	□ TBD	□ TBD	- Start Code ☑ MainMenu, Instructions, DeadScreen, WinScreen
Week 5 (Oct 19 -Oct 23)	□ TBD	□ TBD	☑ Research techniques, protocols, and procedures
Week 6 (Oct 26 -Oct 30)	□ TBD	□ TBD	☑TMX map creation (Oct 27)
Week 7 (Nov 2 -Nov 6)	□ TBD	□ TBD	☑ Sound (Nov 15) PhysicsMain, Graphics(Nov 8)
Week 8 (Nov 9 -Nov 13)	□ TBD	□ TBD	☑ Entity(Nov 22)
Week 9 (Nov 16 -Nov 20)	□ FINISH PROJECT (Nov 23)	□ FINISH PROJECT (Nov 23)	☑TMX map implementation (Nov 14)
Week 10	□ Finish Presentation	□ Finish Presentation (Nov	☑ Finish Presentation (Dec 4)

(Nov 23 -Nov 27)	(Nov 30) - Assign who says what, “Practice” □ Project Due (Dec 4)	30) - Assign who says what, “Practice” □ Project Due (Dec 4)	- Assign who says what, “Practice”
Week 11 (Nov 30-Dec 4)			☑ FINISH PROJECT (Nov 23) ☑ Project Due (Dec 9) ☑ Presentation Due (Dec 9) ☑ Final Report Due (Dec 11)

Conclusions and Future Work

Kunal Babbar: Throughout the project, I ran into some challenges along the way. Some were solved within hours while others took days to resolve. At the beginning of the project, I did not know how we were going to connect all the classes as we had to make one level. As the project progressed, I started to have a better understanding of inheritance and how classes interact with each other to connect different classes. Debugging was also a big problem for me. Little things like adding sound functions and some .mp3 files took longer than I expected. I had to set breakpoints in certain parts of my code to see where the error was coming from. Past projects like TrafficJam gave me a strong starting point in order to get myself ready for bigger projects like Super Mario. For the future, I will help develop more levels as well as adding more enemies to each level. Increasing the length of each level while adding more platforms, coins, question mark blocks are all things I can work on in the future. A piece of advice that I would give to future comp 55 students is to focus on the backend of the game rather than putting a lot of effort into the graphics. Getting the basic functionality of the game is much more important than how the game looks.

Michelle Vu: Some of the technical challenges that I encountered throughout the process of the project consisted of topics that went beyond our knowledge such as the implementation of the TMX library. It was a whole new topic that we had found to design the levels easily. While the process of creating the TMX map level, my teammates and I had to do a lot of research and trial and error to figure how to go about implementing the TIleReader to read in the TMX file. One of the biggest techniques that helped go about the code was to break down the problems and categorize them into small parts to tackle. By doing so, this allows you to focus on one thing at a time so that it isn't too overwhelming. It also helped to be able to talk it out with a teammate and work through it theoretically and logically in the code. I would have liked more knowledge about the starter code so understand how all the classes were connected so that we are able to figure out what is necessary/relevant for our specific project. Being able to experience this project design class, I come to learn many of the important qualities that make up a successful and proactive team. Having a team of open-minded and understanding individuals really helped in providing a comfortable environment for support and growth. Being able to have that in a team, lets all team members have the opportunity to ask for help and questions to further their understanding. I also learned that when tackling new languages there is a lot of personal research that is involved to try figure out how to go about starting. As for my advice for future COMP 55 students, I would advise you all to be open-minded to different ideas, communicate with teammates, and make sure to do your own personal research!

Kaung: One of the biggest challenges during the project is the implementation of PhysicsEngine. There were multiple things that needed to be working at once, such as movement, gravity, collisions and game conditions (winning, losing). The decomposition lab I did earlier in the semester greatly helped out, as it taught me to break down the problem into smaller portions and implement each of them over the period of the project. Another problem I faced is working with code that's not mine. During the current project, I had to work with external libraries such as ACM and TiledReader. I had much more trouble working with TiledReader than with ACM, as although being fairly well documented, examples using the TiledReader library were non-existent, so I had to go through all of the documentation just to get started. I've worked with others' libraries in the past in personal projects, so I'm used to reading documentation which has helped me learn the TiledReader library. Being new to object-oriented programming, I also found it difficult keeping classes cohesive. A good example would be the Entity class and the main class. There are many elements inside the main class that should've been kept in Entity class, such as the sprite sheets and character sprite arrays. In the future versions of this project, we would add power-ups, a larger variety of enemies and collectibles such as coins. We would also experiment with different sprite sheets to create levels with different themes and designs. As for advice, I strongly suggest getting started as soon as possible. NEVER put off tasks because you're not sure what to do, just get started. You will slowly figure things out.

Ade: Knowing how to use JAVA and its army of pointers was the most technically challenging part of developing this game for me. I felt exponentially more comfortable with the language in subsequent weeks, having good teammates played a significant role in allowing me time to grasp different techniques and concepts. The most important software engineering technique I learned was how to break big problems into little problems. Applying it to software development is not immediately intuitive. Tasks can and have gotten away from me because of my inability to start doing the foundational items first. Another technique I learned was to visually conceptualize the code before starting to write it. Sitting down and drawing on a blank sheet of paper is often how I began to add my pieces of code to the project. I am familiar with project management, and this project is not different. There are parallels that allowed myself and the team to stay away from some of the pitfalls associated with group work. This prior knowledge helped me immensely with completing this project. In the future, improving the project would consist of adding power ups and different types of enemies. Additionally, making more challenging levels to mimic the difficulty of the original game would serve as markers for improvement. Adding a theme, such as the recommended COVID theme could eventually advance the game to a place where I would feel like it was a fully completed project. My advice to any future COMP 55 student would be to avoid procrastination with any issues, tackle them immediately, and share progress and or setbacks with your teammates in a timely fashion. This ensures quick help from your faculty/TA/teammate. This leads to projects being completed on time. This is the way.

Murad: While creating the Super Mario Game I had many technical challenges whether it was just getting started and finding the beginning of the code or importing things such as the level. After my team and I were able to figure out exactly where to start we had to be able to get a level on the screen. After many different meetings we originally thought about putting every image for the level in an array list. One thing that really helped us out getting away from that was the TMXTiledMaps. Although it was very easy to use and create a map on, I really struggled in learning how to import the TMXLibrary into eclipse and being able to actually load the level. It became a whole new library that I had to learn from the beginning. After numerous articles and trial/error we were able to somewhat find a path to work from. The main thing that helped me understand the library was just taking it step by step rather than trying to code it all at once. This allowed me to slowly ease into the new library for the TMXReader and get a better grasp on the new things I was learning. I believe that if I had a better grasp on Java in general I would have been able to get things done a little faster. Working with group members and being able to show eachother code made it a little easier since everyone was able to work and look at the same file. In the future we would take this game to the next level adding more intricate obstacles/levels, different enemies, point system, and power ups such as the mushrooms. We had originally planned to do many of these things and even add a Covid theme to the game but had to back track due to time constraints. Advice for future students would be to get started as fast as possible and always be open minded to new libraries and techniques. You have to always be able to work and learn new things so being stubborn is not the way to go. It'll take time and a lot of your own research to understand many of these new programming techniques we learned during the completion of this project. Always remember to stay positive and focus on the work getting done in small pieces, don't try and tackle the entire game at once but rather piece by piece. Just getting something on the screen or a screen design is a great way to start and get working from.

References with Annotations

INFORMATIONAL

a. Super Mario Game
https://supermariobros.io/

i. This is an online SuperMario game which we used to see how an original Mario game is supposed to work. It helped us replicate the movements and working tree of this game to an extent.

b. (Video) Super Mario Bros Gameplay:
https://www.youtube.com/watch?v=rLI9XBg7wSs

i. Due to the fact that our team chose to recreate the well-established, classic game of Mario, it would make sense to review the general controls and essential features that make this game so iconic. By reviewing the overall gameplay, we will be able to translate the key controls as well as put our own spin to the idea to add originality.

TECHNICAL

c. Java Inheritance
https://www.w3schools.com/java/java_inheritance.asp

i. In JAVA, it is possible to inherit attributes and methods from one class to another. This helped us to gain knowledge about the superclasses and subclasses in JAVA.

d. Scrolling Camera:
https://gamedev.stackexchange.com/questions/44256/how-to-add-a-scrolling-camera-to-a-2d-java-game

i. One of the essential features of running the game is the camera movement. This link would supply us with starter points to how we would go about implementing this feature into our code given that this is our first time having to add the element of a camera viewpoint and a moving stage/level.

e. Game Loop Efficiency:
https://gamedev.stackexchange.com/questions/160329/java-game-loop-efficiency

i. Our game uses physics and input polling that requires our game to run at a fixed frame rate, so we were deciding whether we should run the whole game at a fixed frame rate or run the physics and input polling separately, which would add complexity.

f. Keyboard Input:
https://stackoverflow.com/questions/55714167/key-listener-with-acm-graphics

    i. This will provide us some guidance on how to incorporate the keyboard inputs to correlate to the game controls such as start game, jump (up), run(forward), etc.

g. Camera movement
https://pressstart.vip/tutorials/2019/11/29/105/mario-2d-camera.html

    i. This should provide us with an idea of how we should handle the camera movement for our game

h. TMX Tile Map (Video)
https://www.youtube.com/watch?time_continue=1&v=WRS9SC0i0oc&feature=emb_logo

    i. We used this as one of the major references for creating the tile maps for our game. This helped us set a whole map in one go rather than creating separate pictures and then linking them together to create a map.

i. TiledReader Library
http://www.alexheyman.org/tiledreader/

    i. This helped us run our tile maps for the game. It is a simple Java library for reading information from files created with the map editor Tiled. TiledReader makes the internal structure of Tiled map, tileset, and template files easy to navigate.

j. Image Rotation in Java
https://blog.idrsolutions.com/2019/05/image-rotation-in-java/

    i. We used this as reference for our dead screen earlier as we wanted to show an inverted image for Mario representing that he is dead but instead we ended up using a .gif image with a rotate function to move it around its axis.