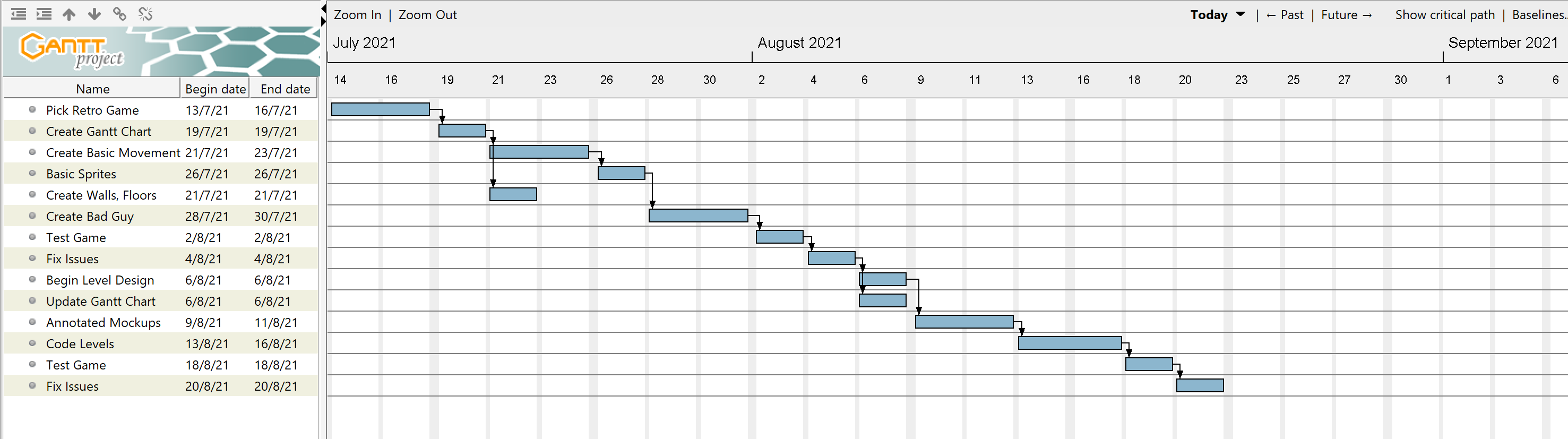
Journal

Gantt Chart 

The Gantt chart shows all the tasks that need to be created for the project to be created, on time. It runs some tasks simultaneously, because multiple people in our group can work on the different tasks. It also only runs tasks on Monday, Wednesday, and Friday as these are the days that we have Computing Class.

Milestones: The first Test Game is a key milestone, along with the finishing of annotated mockups

Slack Time: Since the events only run Monday, Wednesday and Friday, this gives us plenty of slack time, between 1-2 days either side in order to complete a task as homework if it isn’t done within the allocated time.

Changes: We completed our tasks up to begin level design quicker than expected, however this enabled us to use more time for the more important level creation and finishing touches, as these were a lot harder

**Creative Thinking**

**Creating a GitHub Repository** – Using our previous storage medium of OneDrive created a problem with two or more members working on the same project simultaneously. Because of this, we found a different storage medium in which merging different files can be more streamline and the files are still stored online so anyone can access it.

**Creating a GUI-** An appropriate font could not be found for the GUI, so it was decided to use a case statement, using sprites to create an accurate looking GUI that fit the look. This used a statement for the 10’s and the 1’s, tracking the variables and displaying them on the screen.

**Controls-** The original Metroid II was built for the Gameboy, meaning that we had no reference to go off for controls. We decided to use arrow keys for movement, as this is most common in most games and feels the most natural.

**Sprites-** Samus would perform tasks such as shooting a missile or rolling, but the sprites would not match this. To fix this, we had to find many sprites that fit all the states that Samus could be in, and all of their actions.

**Color­-** Since the original Metroid II was made for the Gameboy, it was completely in black and white. We considered transferring the game to colour, to give it a more modern feel however we decided to stay with the black and white to stay true to the original game and the artwork.

**Progression in difficulty-** To make the game grow in difficulty, we designed the levels so that they start out easy, before getting harder and harder. This allows people to understand the mechanics of the game before progressing into harder levels that challenges them more.

**Critical Thinking**

**Merging conflicts**- While using GitHub, we struggled to combine changes from multiple users working simultaneously. This often led to code being deleted, and work being overwrote. We eventually figured out how to use merging to keep all the changes without losing important code.

**Hornoad movement-** When creating the Hornoad, the AI kept running into issues getting stuck in the wall and not jumping. Using trial and error, we found code that worked around the collision problem, making the Hornoad turn around when it hits a wall

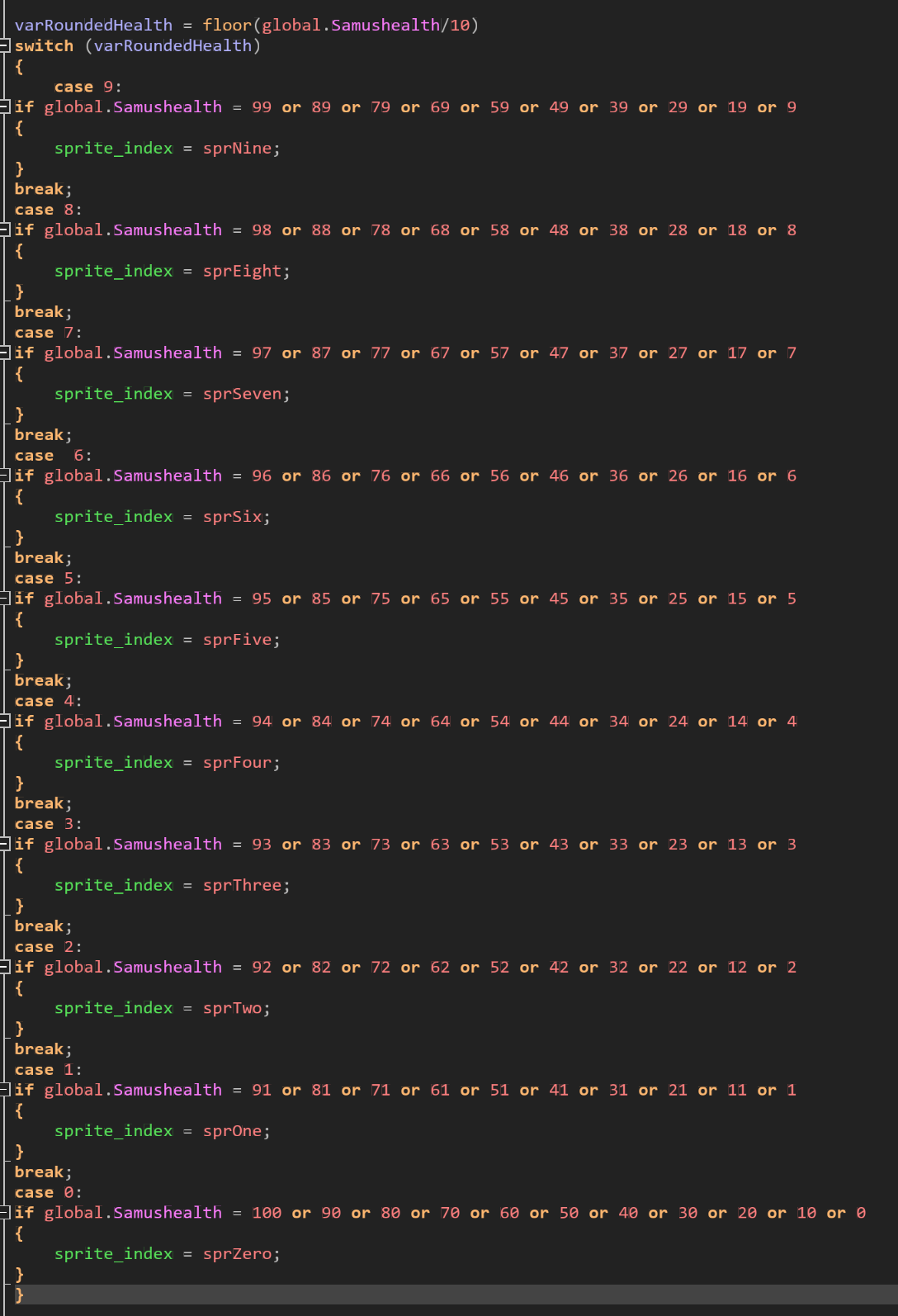
**Samus’s health & iFrames** – When Samus would collide with an enemy, he would lose all her health very quickly as the code ran in the step event and would drain her health. To fix this, we used an alarm that prevented Samus from taking damage within 60 frames of taking damage originally

**Samus’s sprites moving**- When Samus would change sprites from standing, to shooting and more his origin and hitbox would change, meaning he would clip into the ground. In order to fix this, we had to go through all the sprites and manually align the origins so that they stay centered throughout all of Samus’s states, not just the one sprite

**Room size vs Resolution:** Originally, we planned to use a room size of around 5000x5000 pixels. However, Samus and other sprites looked much smaller than expected. When we attempted to enlarge them the sprites became quite low resolution so we had to find a perfect room size that would let us build a good size room, while having high resolution sprites.

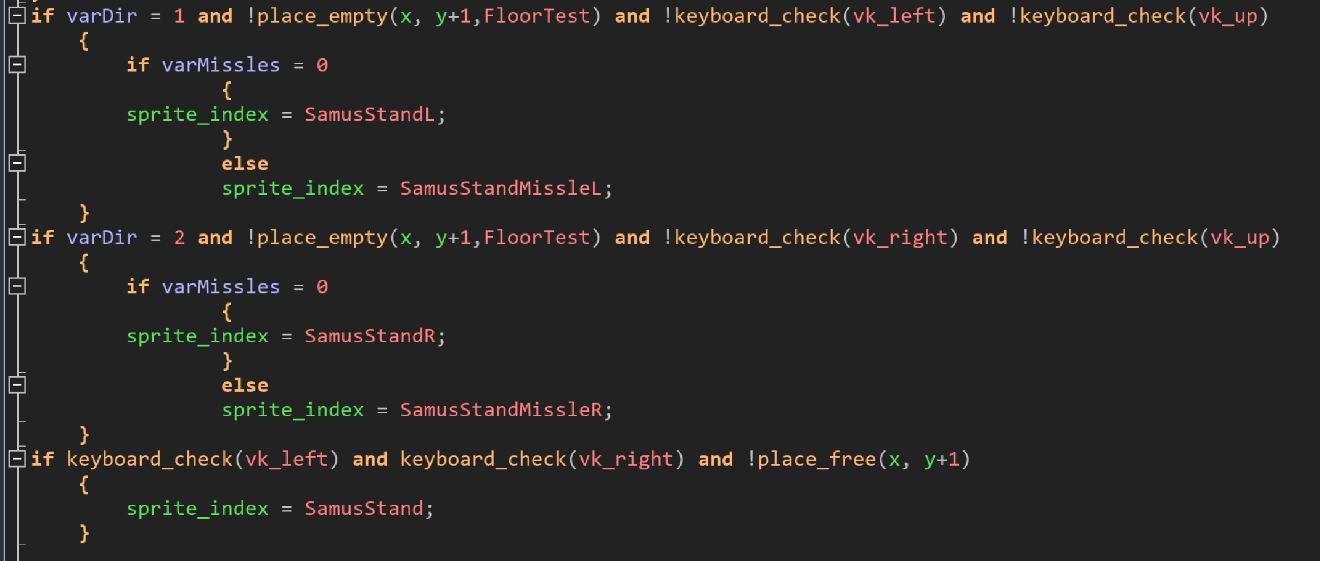
**One room vs Many rooms:** We were stuck between using one large room, or many smaller rooms, for our game design. While one large room would be easier, and smoother gameplay, it could also lead to a lagging game, and make it harder for multiple people to work simultaneously. Many smaller rooms on the other hand would give us more control, however game play would be jumpier from room to room, making it harder.

**Three New Skills**

**Case statements for GUI**

An appropriate font could not be found for the GUI, so it was decided to use sprites instead. This meant that a case statement had to be created, to effectively track the health and ammo left. There are 2 case statements for each variable present on the GUI, one for the 10’s, and one for the 1’s. The left shows the code for the 10’s for Samus’s health.

1. **States**



Samus has multiple different states, including standing, crouching, jumping, ball, shooting and many more. If we just used key pressed events, these states would collide leading to glitches. In order to prevent this, we created checks for each one of Samus’s states to track if they are crouching and depending on which key is pressed or other factors, to appropriately change Samus.

**Functional Requirements:**

* **Game runs as expected:** The game launches without any errors and runs at a smooth frame rate.
* **Working movement and collisions:** Samus can move around properly without getting stuck in objects, it all works as expected
* **Enemy AI works properly and efficiently:** The enemy will do what is expected, either tracking to the player or walking in a loop to create a challenge for the player
* **GUI properly represents info:** The variables on the GUI are accurate, in order to show the player the proper info required.
* **Rooms transfer properly**: Once the player has completed a room, he can move on to the next section to continue
* **Buttons work:** The menu buttons interact properly so that the game can be started, etc.
* **Death:** When Samus or an enemy runs out of health, they are removed and if Samus dies, a death screen is showed

**Non-Functional Requirements:**

* **Retro Feel:** As we are creating a retro game on a modern software, it is important to capture the older feel of the game, through things such as a square game window.
* **Easy to understand, User-Friendly**: The game makes sense and can be played by a person with no prior experience
* **Good looking sprites**: The sprites are high resolution, detailed and portray an idea clearly to create a proper game
* **Sounds that match the game:** Sounds for things such as shooting, collecting an item should match the event to build on the game’s immersion
* **Progression in difficulty**: The game starts off easy, while building up in difficulty to create a proper difficulty curve and make a challenge.
* **High score tracking**: The high scores of the players can be tracked so that people can compare scores, and see how well they did
* **Clear start and end screens:** A start and end screen help build on the overall look and feel of the game, making it feel much more polished and completed.

**Constraints**

* **Economic:** We have no budget for game production, so we are limited to free resources for things such as sprites and sounds
* **Time:** The game and all other elements must be completed within 7 weeks, adding time pressure
* **Multiple people working**: As we have 3 people in our group, who often all have to code at the same time, work arounds had to be found to use the GMS project at the same time.
* **Software:** We can only use Game maker Studio, which while a good software still has some limitations and issues present.
* **Skill levels:** We are all at different levels of programming and design skills, so it was important to delegate tasks appropriately for each person.
* **COVID-19**: Lockdown has forced us to all work at home, meaning a lot of design is harder, and we have had to work online and find ways to digitally collaborate.