



CAVIO
HOSSAIN

A-LEVEL
CS NEA

RPG ADVENTURE JAVASCRIPT GAME

JS

cavio.hossain@laetottenham.ac.uk
By Cavio
Candidate No: 2002

2022/2023

Contents Page

ANALYSIS	3
Problem Description	3
Similar Solutions	4
Computational Methods	7
<i>Collision Detection</i>	7
<i>Animation</i>	8
<i>Game Loop</i>	9
Stakeholders	10
<i>Stakeholder Survey</i>	11
<i>Choosing Stakeholders</i>	13
<i>Stakeholders Roles</i>	14
<i>Stakeholders Interview</i>	16
Feasibility	19
<i>Managing Time</i>	19
Languages, IDEs and Softwares	22
<i>Javascript & Replit</i>	22
<i>Softwares</i>	23
Hardware & Software Requirements	25
Limitations	27
Success Criteria & Requirements	28

DESIGN	32
Objectives	32
<i>Game Design survey</i>	32
GUI	33
<i>Layout GUI</i>	34
<i>Menu GUI</i>	35
<i>Dialogue GUI</i>	36
<i>Battle GUI</i>	38
Software Design	40
<i>Decomposition</i>	40
<i>Object Oriented Programming</i>	40
Game Class	41
<i>Game Loop Algorithm</i>	43
<i>Game Loop Flowchart</i>	44
<i>Collision Detection Algorithm</i>	45
Entity Class	47
<i>Sprite class</i>	49
<i>Animation Movement Algorithm</i>	50
Player Class	52
<i>Controlled Movement Flowchart</i>	54
<i>Controlled Movement Algorithm</i>	56
<i>Controlled Movement Usability</i>	58
Game Battle	59
<i>Game Battle Flowchart</i>	60
Unified Modelling Language	61
Testing	63
<i>Post Development</i>	66
Structure of Solution Development	68

DEVELOPMENT	69
Canvas Setup	69
<i>Canvas Style</i>	70
Game Class	76
<i>Drawing Layers</i>	80
<i>Game Loop</i>	89
Map Class	93
Entities	102
Player Movement	119
<i>Tiled Based Movement</i>	120
<i>Controlled Movement</i>	123
<i>Speed Boost</i>	131
<i>Animation Movement</i>	146
<i>Character Camera</i>	162
Character Transformation	168
<i>HUD</i>	174
<i>Effects</i>	181
Collisions	192
<i>Collision Detection</i>	193
<i>Working Collisions</i>	199
Map Navigation	204
<i>Map Transition</i>	209
NPC Behaviour	215
<i>Movement and Collisions</i>	215
<i>Behaviour Loops</i>	237
<i>Player Interaction</i>	249
<i>Dialogues</i>	258
Evaluation	265
Success Criteria	265
Usability Features	269
Stakeholder Testing	272
Functional Testing	273
Robustness Testing	274
Limitations & Issues	275
Maintenance	278
Further Development	279
Final Code	280

ANALYSIS

Problem Description

There have been substantial technological improvements in each generation, and each advancement has allowed for a quicker and greater growth of technology for the next generation; as a result, technology has increased exponentially.

"To see this, imagine making a chair with hand tools, power tools, and finally assembly lines. Production gets faster after each step" - Berman, Alison E.

The evolution of video games is directly correlated to technological advancements, as graphics technology has become increasingly complex, from 2D to 3D, VR etc.

However, one of the reasons why this is an issue, is that it has resulted in the under production of retro games and an 'extinction' of simpler games such as pixelated games, as video games technology is continuously evolving. "The days of pixelated screens and limited sounds are a distant memory as video games have become more lifelike than ever. Video game creation has become increasingly complex" - Beattie, Andrew.

Another reason why technological advancements in game graphics is an issue, is that games with high quality realistic graphics are more prevalent nowadays and they require high-end computers and additional hardware, so players can play with a decent performance, but not everyone can afford high-end devices, as technology advances, the price of consoles increases proportionally.

Therefore, a pixelated 2D web game will be ideal and affordable for every console, whether it is low-end and high-end computers and even suitable for mobile devices, as pixelated games are usually vastly compatible with every type of hardware regardless of its CPU, RAM, GPU, and storage

In conclusion, I have decided to code a 2D pixelated game to grasp that retro feel that is missing from video games nowadays and allow compatibility for most devices without requirements of expensive hardware and specifications.

After some careful analysis on deciding what game or project would be suitable and would maximise the enjoyment of players, I've chosen the following:

2D RPG Adventure Game. I've specifically chosen this as I am under the impression that a lot of features can be implemented alongside the RPG game, which usually involves top-down character movement, maps & locations, special abilities, and NPC interactions. However, the goal of the game is really simple, which is to **explore** by discovering different maps and interacting with different NPCs

Similar Solutions

Pokemon

One of the most successful pixelated 2D adventures game is “Pokemon Firedred” which is part of the Pokemon franchise, where the player travels a particular region with the aim of catching creatures called “Pokemon”, the player faces challenges, like battling against other trainers or their rival throughout their gameplay.

The game does not have an end, as the player can either explore further regions by continuing with the game story or wander around in towns and routes by catching more creatures, exploring different maps and interacting with different NPCs.



Moreover, the battle system is also a great addition, in this existing solution, the user is able to battle other pokemon trainers in a turn-based battle system, which is another similarity that I am willing to implement. Also, the player movement and animation in this game is really simplistic which persuades me to follow this existing game's approach to solve the character movement problem, where the player can only travel 4 directions, respectively to the user's input.

Finally, the speed boost feature will also be a good feature that will allow the player to travel faster, this feature is successfully executed in Pokemon FireRed, when the player receives a pair of running shoes, however, I am going to implement this feature but the player can only toggle their speed boost when they toggle on their special ability.

I could not integrate the speed boost with the special ability feature such as in the Pokemon game, which may increase the overall complexity of my proposed solution because the player can now use two different actions instead of one, and increase the game mechanics which may appeal to many gamers. However, integrating the speed boost with the special ability is the better approach in this case because it maintains the simplicity of my game, also it will create a sense of synergy between these two distinct features and may enhance the overall gameplay.

Stardew Valley



Stardew valley is a popular game that focuses on farming and emphasises the exploration aspect of RPG games. There are many attractive features that I am going to implement in my own game, such as the customisable player skins, NPC dialogues, interactions and behaviour.

The NPC behaviours and interactions will also be a great addition to the development process of my game, where I will follow a similar approach to that of the Stardew Valley franchise, where different NPCs behave in different behavioural loops, have different dialogues, sprites.

Unlike other RPG games such as Pokemon or Zelda, this game displays the NPC's faceset when the user interacts with the NPC as seen in this picture. Which I am definitely going to add in my game.



I could not add a faceset and follow a different layout for the NPC interaction dialogue, as the current approach may lead to some technical limitations as the game is required to fetch more images and execute more requests and instructions than it has to, slowing down the performance of my proposed solution. However, I am inclined to add this common feature from Stardew Valley because it helps to establish distinct personalities and give NPCs more character, which will make the game more engaging and dynamic by enhancing immersion as it makes the game feel more alive.

There are some differences with my proposed solution, my game focuses on being more action-oriented by including features such as special abilities, speed boost and battle system. Whereas Stardew Valley is more 'pacific' as the game is based on farming and agriculture rather than a fantasy setting.

Legend of Zelda



Legends of Zelda, specifically the older generations such as '**Link to the past**', is a classic game which focuses on combat and especially exploration. Like most other solutions, this game also follows the common top-down view approach.

I admire that simplicity and the pixelated theme in this game, as well as the fantasy setting. Moreover, this existing solution uses **Heads-up display (HUDs)**, which are visual cues that display important information such as level, health, inventory on screen.

However, in addition to the graphical qualities, there will be some significantly more complicated features that are not generally included in Zelda games, such as special abilities during the open-world game experience. Furthermore, my game will have a true open-world architecture, allowing the user to explore diverse landscapes far more than the Zelda games do. Finally, another difference between this existing solution is that the Zelda game only features one character sprite, giving the user limited choice in customizability, therefore I am going to avoid this issue and let the player switch between a variety of character sprites.

I could only restrict the player to use only one character sprite like in the zelda game, which makes the game more consistent and creates more immersion. However, it may depersonalise the character and encourage player engagement.

Computational Methods

Collision Detection

Collisions are the interactions that occur between the player and the objects, walls, and beings that come into contact with each other. This computational method is an essential element because it allows the player to interact with the map's items and sets boundaries for moving entities intersecting with the map and each other. Without this crucial component, the game will be incomplete due to a lack of boundaries.

For example, a player entering a tile occupied by a wall or tree will be prevented from travelling further in that direction. Another type of collision might happen when the player interacts with other entities on the map, allowing the game to activate NPC dialogues or cutscenes. Without collisions, the player could move around anywhere they wanted, which would remove the purpose of having maps in the game.

To sum up, collisions are really important in most games, especially top-down 2D games like mine, and this feature will allow the player to interact and engage with the maps and entities in a meaningful way.

I could add overlapping bounding boxes instead of using pixel-perfect collisions. This approach can add boundaries between objects and the player by detecting when these two entities or objects intersect. Moreover, this approach might be better because it uses less resources and code to implement it, also it may increase performance too and will allow the player to pass through narrow gaps instead of being stopped for every pixel.

However, I am sticking to my original approach because of the tile based movement that I have implemented previously and pixel-perfect collision is easier to implement than overlapping bounding boxes, meaning my original approach will save me more time.



Moreover, collisions are necessary if I want to implement other features such as NPC interactions. The player is required to contact the NPC which will allow them to engage with that NPC. This process will definitely require a collision detection system so the game is notified when the player interacts with an NPC. This is a common approach in existing solutions such as in Pokemon FireRed.

Animation

Animations are essential for a top-down 2D RPG game. It creates an immersive user experience by giving the game a sense of motion. This computational method will be used for features such as character movement, special abilities and effects. I will be using a frame-by-frame animation.

Frame-by-Frame Animation

This approach involves an algorithm which creates animations using a sprite sheet containing a series of individual frames, where each frame represents a certain direction or pose for the animation progress. When the frames are displayed rapidly using a programming construct called a game loop, it will create an illusion of motion, the animation will appear smooth and seamless.

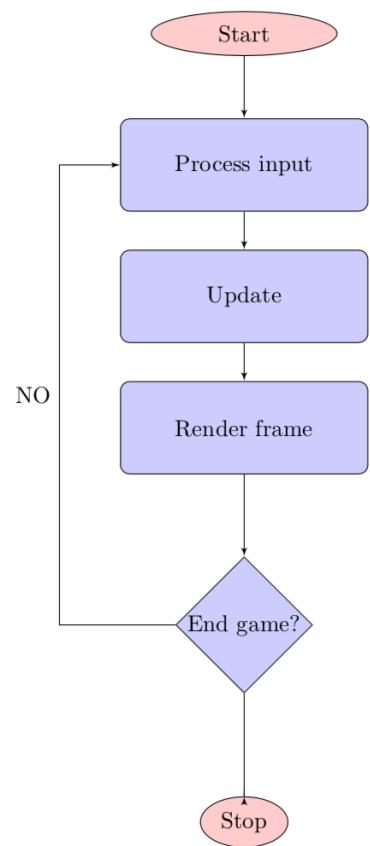
For example, if I wanted the character to travel in the right direction, the game would select the 'walk-right' column from the character's spritesheet which would contain about 4 frames. These frames will be stored in a list and while the character is walking, the game will iterate through that list and set the frame as the character's current frame for every iteration. This process gives an illusion of motion when the character walks to the right.



Additionally, the direction and movement animations for the player's character will be dependent on the user's input, this is similar to most games such as The Legend of Zelda as seen in the example.

I could use a different approach such as skeletal animation which involves defining a hierarchical structure of bones for the character, and configuring the bones directly to generate the desired animations. This approach is more efficient and allows for more flexibility compared to the frame-by-frame animation approach. However, skeletal animation is definitely not suitable in this case, especially for 2D games like mine. Moreover, skeletal animation will not be a good approach as the animation will end up looking less natural, smooth and fluid.

Game Loop



A game loop is an essential computational method, mainly used in video games, with the purpose of updating the game's state whether there is a change to the game or not. The game loop runs continuously as long as the game is running, where it executes a series of instructions, as seen in the diagram. Some of the instructions that the game loop will handle are drawing images, detecting user input, and updating the game's state. This computational method will allow me to solve more problems for the proposed solution as it allows me to create an interactive game that responds to the player's action in real time.

The main problems that the game loops can solve:

1. Detecting user input in real-time; This allows the user to interact with the game. When the user inputs a key, the game loop will respond accordingly, committing changes to the game's state. This will allow for the development of other components in this game such as player movement, speed toggle, special abilities. Collision detection, entity interactions etc.

2. Updating game state; Throughout each interaction in a game loop, the game state is updated. One of the main purposes to update game state is to allow different entities to update their state and position.

3. Rendering images and animations: The game loop will ensure that the images are rendered and drawn correctly onto the canvas, this also allows the development of animations, where through each iteration in a game loop, the game state is updated constantly, therefore different images are constantly being drawn creating an animation effect.

I could just use a static game, which involves no game loop, this will save more time and shorten the length of the code and resources used for this project. However, the game loop will be necessary especially for updating the game state, otherwise, the user won't be able to interact with the game at all as the game won't respond to any input from the user because it won't be able to detect it.

Stakeholders

In order to broaden the well-being of my project, I've decided to keep in contact with stakeholders which can provide valuable insight and information regarding my project. The main group of stakeholders that I will be communicating with are the following:

1. Casual gamers
2. Competitive gamers
3. RPG Gamers
4. Game Developers

It is important to communicate and gather information from stakeholders in order to broaden the aspects of my project and understand where to allocate and invest my time and resources in the project.

However, it is sometimes difficult to communicate and gather reliable information, which all depends on which stakeholder should be trusted.

The screenshot shows a Google Form titled "Stakeholder Summary". The form includes a descriptive text about the purpose of the form, a required field for the person completing the form, and a classification section for stakeholders. The classification section contains two options: "Individual" and "Organization".

We will use this form to enter initial information about stakeholders relevant to our work.

* Required

Who are you? (Person completing form) *

Your answer

Stakeholder Classification *

Individual

Organization

Therefore, I will need to carefully analyse what stakeholders are the most reliable when giving feedback, this can be done by surveying potential stakeholders.

I plan to use google forms to create a survey which can be used to dictate who should be a stakeholder for this project.

Example of a stakeholder google form survey

I have chosen google forms in particular because it is easy and simple to use and create surveys, cross-platform and accessible to everyone who has the link.

I've chosen gamers and developers as they all share a common interest in games, as there are a different variety of gamers playing different genres of games.

I will need to perceive which stakeholders specifically invest more time playing 2D pixelated adventure/action games, this would be the RPG gamers.

The players will be the stakeholders who provide feedback about what part of the game needs improvement or adjusting suitable for usual RPGs.

I could not use stakeholders because it can be quite difficult to find a good amount of stakeholders, as the lack of stakeholders would offer limited perspectives on the project and it takes time to find appropriate stakeholders. One of the methods I think would be suitable to gather a diverse large number is to use a Google Form survey, as the links can be shared easily and posted in online forums so people with similar interests in RPG games can take part in the survey. As a result, the survey will produce a good number of responses from potential stakeholders. This process will significantly make finding stakeholders easier and faster, saving me more time.

Stakeholder Survey

As mentioned previously, I will be using google forms to gather information about stakeholders which will help me consider which stakeholders to choose from based on the results and answers. Here's some of the main questions included in the google form.

Target audience

Which one of these describes you the best?

- Casual Player
- Competitive Player
- RPG Player
- Game Developer
- None of the above

As mentioned previously, I will be needing to identify what relations the potential stakeholder holds towards gaming and especially RPG games. I will need a diverse range of stakeholders so I can cover all of their advice about things to add, improve or remove from the game, which broadens the perspective of the game.

Knowing how many hours a stakeholder plays video-games a week will help me identify regular video game players, which will allow me to deduce who to pick as a stakeholder as some candidates will potentially have more knowledge about good user experience etc.

Estimate how many hours do you play video-games a week

- 0-1 Hours
- 1-3 Hours
- 3-7 Hours
- 7+ Hours

Occupation

What is your occupation

Your answer

If employed, how many hours a week do you work?

- 6 - 10 Hours
- 11 - 19 Hours
- 20 - 30 Hours
- 30+ Hours
- Zero Hour Contract

The following question will dictate how busy a stakeholder is.

It is important to consider their working hours; A stakeholder, which is a teacher, will get emails from other students too, which could delay their input to the project.

A stakeholder which is an investment banker will most likely not have any free time to contribute ideas to the project. A stakeholder which is a part-time worker will have more free time to input their ideas and feedback in the project.

Communication

What are your preferred way of communicating? *

Please select multiple if possible.

- Telephone Calls
- Email
- Live Meetings
- Online Meetings
- Mobile Messages
- Social Media Messages
- Discord
- LinkedIn

I have also included a section where the potential stakeholders can select their preferred way of communicating and provide their contact details.

Enter contact details for the communication methods

Your answer

Choosing Stakeholders

After receiving all the responses from my google form survey, I have decided to choose the following stakeholders for my project. I believe they will be effective and beneficial to receive feedback from these stakeholders to make my game more engaging, as I can receive useful feedback on what to add and what to change while developing and designing my game.

Name	Age Group	Type of player	Occupation / Hobby	How many hours do they work in a week?	Communication Method
Prince Camayah	13 - 17	Competitive Gamer Game Developer	Student	30+ Hours	Email
Muhaimin Ahmed	18-24	None of the above	Student Graphic Designer Pixel Artist	20-30 Hours	Email Discord
Wessel Broek	25 - 34	Casual Gamer RPG Gamer	Audio-Engineer	Zero Hour Contract	Mobile Messages Email
Ivo Igor Kucher	18 - 24	Casual Gamer RPG Gamer	Customer Support	30+ Hours	Discord
Andre Billey	13 - 17	Casual Gamer	Student	30+ Hours	Email Mobile Messages
Bartosz Bester	18-24	RPG Gamer Game Developer	Indie Game Developer	11 - 19 Hours	Telephone Calls Mobile Messages Online Meetings Discord

Stakeholders Roles

Prince Camayah: As a competitive gamer and game developer, he can provide valuable feedback on gameplay mechanics and level design. He may also have insights into popular trends in the gaming industry that could inform your game's direction. His role could include providing regular feedback via email and providing occasional consultations on game design. I could not choose him as a stakeholder because he works 30+ hours a week, however, he is important for this project because he can ensure that the game's mechanics are balanced and enjoyable for players.

Muhaimin Ahmed: As a graphic designer and pixel artist, this stakeholder can provide feedback on the game's art direction and graphics. He may also have insights into popular trends in pixel art and how to create visually appealing graphics for the game. His role could include providing regular feedback via email and Discord, and advising on implementing pixel art and graphics for the game. I could not choose him as a stakeholder because he's neither a gamer or a game developer meaning that he may not be able to provide more information on the actual gaming experience, however, this stakeholder is important because the graphics and visual design of the game are crucial for creating an immersive and engaging player experience.

Wessel Broek: As a casual and RPG gamer with an audio engineering background, Wessel Broek can provide feedback on the game's sound design and music. His interaction and role could include providing regular feedback via mobile messages and email, participating in playtesting sessions, and advising and implementing sound effects and music for the game. I could not add this stakeholder because I may not be able to add any sounds and music in this game because of its complexity, however, this stakeholder may be a good choice because sound design and effects can greatly enhance the game's overall atmosphere and player experience.

Ivo Igor Kucher: As a casual and RPG gamer with customer support experience, Ivo Igor Kucher can provide feedback on the game's user interface and user experience. He may also have insights into common issues players encounter and how to resolve them. His role could include providing regular feedback via Discord, participating in playtesting sessions, and helping to troubleshoot user issues. I could not choose this stakeholder because the game will not require a complex user interface, whereas the user interface is the main purpose of why I chose this stakeholder to provide feedback and advice on, therefore, he may not be necessary for this project. However, he may offer more value on implementing a good user interface and experience, enhancing the player's experience and reducing frustration.

Andre Billey: As a casual gamer and student, Andre Billey can provide feedback on the game's accessibility and ease of use. His responsibilities could include providing regular feedback via email and mobile texts, participating in playtesting sessions, and suggesting ways to make the game more accessible to a wider audience. I was unable to include this stakeholder in my project since I already have other stakeholders who have provided similar feedback; yet, he is significant because a game that is simple to use and comprehend is more likely to appeal to a wider audience and be successful. Furthermore, similar stakeholders may have different approaches and advice to offer because the google form survey had limited questions and did not test their knowledge of the game development industry.

Bartosz Bester: As an RPG gamer and an indie game developer, Bartosz Bester can provide feedback on the game's design and programming. His role could include providing regular feedback via telephone calls, mobile messages, and online meetings, participating in playtesting sessions, and offering consultations on game development best practices. I could not add him to my stakeholder team and look for a more experienced and older game developer, who is in a higher age group, however, this stakeholder might be the best one so far as Bartosz focuses on indie games, which are independent game, and since I am working on my project independently, he may be able to help me with this project as my game counts as an indie game. Furthermore, his availability is flexible as he only works about 11-19 hours per week, he enjoys RPG games as well meaning that he has experience in this field, and finally, he offers a wide range of communication methods.

Stakeholders Interview

Prince Camayah

What feature do you think is missing from most browser-based RPG games?

Answer: "The lack of GUI and actions that can be performed by the user, which are really limited in RPG and indie games nowadays"

What's your favourite RPG game of all time, and why?

"I don't play RPG but if minecraft counts then minecraft, because the creativity allows me to explore some different concepts and create anything I desire, meaning there are no limits and I really enjoy the open-world feature and the freedom of this game."

Muhaimin Ahmed

What inspired you to pursue pixel art and graphic design?

Answer: "I've always liked old game aesthetics, thus I hope to see more games featuring pixelated visuals become popular."

What is the most significant part of game design and art?

Answer: "The most important aspect of game art and design is creating a consistent and immersive world in which players may become lost."

How would you approach designing the characters for an RPG game?

Answer: "My approach to character design would be to use distinct and memorable spritesheets with distinct personalities, quirks, and backstories."

Wessel Broek

What do you enjoy most about your favourite RPG?

Answer: "My favourite RPG game is The Elder Scrolls V: Skyrim due to its massive open world, exciting gameplay, and endless replayability," says the answer.

How would you go about using sound effects for our RPG game?

Answer: "I would approach sound effects design by first analysing the several types of sounds required for the game, such as ambient, dialogue, and fighting sounds." Next I would gather or produce the necessary sounds and integrate them in a cohesive and immersive manner. I would also make certain that the noises are well-balanced and not overpowering, and that they provide useful feedback to the user."

Ivo Igor Kucher

How would you respond to user feedback and concerns in our RPG game?

Answer: "I would respond to player feedback and concerns by actively listening to them, providing clear and useful responses, and working with the team to resolve any issues as soon as feasible."

How do you believe user feedback can assist the RPG game?

"I believe that user feedback is critical for detecting and resolving bugs, improving gaming mechanics, and making the game more entertaining for a broader audience."

Andre Billey

What is your favourite aspect of playing casual games?

Answer: "My favourite aspect of casual games is the flexibility and simplicity to play them for little amounts of time without committing to a lengthy gaming session"

What feature do you believe is lacking in most casual games?

Answer: "I believe that most casual games lack replayability and depth, making them feel repetitive and dull after a time."

What ideas do you have for making our RPG game more appealing to casual players?

Answer: "I believe we can make the game more accessible by giving clear tutorials and instructions, as well as additional difficulty settings and gaming styles that fit different skill levels and tastes"

Bartosz Bester

What do you think is the most crucial part of game development?

Answer: "I believe that the most important aspect of game creation is offering gamers an entertaining experience while also paying attention to detail, balance, and polish."

How would you approach creating innovative and compelling game mechanics for our RPG game?

"I would approach game mechanics design by brainstorming and iterating on new and exciting concepts, playtesting and balancing them, and incorporating team and user input."

After analysing the comments from each stakeholder, I can determine that I have a fair range of stakeholders who can provide varied input and feedback, which will be useful in offering valuable insights into the user's various viewpoints and interests. Every stakeholders' interests have been captured, from competitive players to casual gamers, and from game creators to audio experts.

Stakeholder availability may have an impact on the project to some level, however they have been upfront about their availability, and their different stakeholders have different availability hours, making it easier to reach stakeholders.

Most stakeholders have agreed to be available until the project is completed, while some have noted that their availability may be limited owing to other obligations. I have chosen these stakeholders carefully because I wanted to make sure that each stakeholder has the expertise required to assist with the design of the game, and ensured they all have different degrees of knowledge so I can extend the range of different perspectives.

I could limit the amount of stakeholders to help me speed up the decision-making process and eliminate communication overhead, as fewer stakeholders may result in faster decision-making and shorter duration of the project so I can keep on track with the project.

Also, having a large number of stakeholders may cause communication difficulties because more time and effort is required to manage them all. On the other hand, however, having multiple stakeholders will also increases the diversity and variety of perspectives from the stakeholders which is vital for ensuring that the game is intended to satisfy a wide range of players. As a result, this increases engagement from the future users. Moreover, there is a better chance of receiving creative, innovative ideas and feedback that would have been not conceived if I did not include more stakeholders. The participation of stakeholders with specific expertise such as audio engineers and game developers can assist the development of the game and create a successful well-received game.

Feasibility

Coding this project will be a difficult challenge as I am undertaking a new sector of programming. Therefore, it would most likely involve an abundance of learning and I will certainly encounter many errors throughout the development process. For this project I am going to use Javascript. The game will be played in a web-browser rather than downloading additional software. I could use a different programming language such as Python, which I am more familiar with, provides easy syntax and is easy for me to code in this language which could speed up the development process. The major reason I chose Javascript is because of its versatility and support for visual effects. I could have used Pygame instead, which is part of a Python library designed expressly for developing games like mine, but because it is built on top of a low-level library, it would surely influence the game's performance, resulting in slower and glitchy gameplay.

Overall, I don't want my game to be only accessible as a desktop application, because it will require direct access to hardware and the game's performance will depend on the user's hardware, the main purpose of this game is to be accessible and enjoyable to everyone regardless of their hardware and device's capabilities, therefore using Pygame will be counterintuitive as it is not solving the problem with the proposed solution.

Managing Time

My solution to retain the quality and standards of my NEA without being affected by other priorities is managing my time effectively using planning tools to illustrate my project schedules and efficiently allocate time and resources.

One solution to this approach is using Gantt Charts, which is a graphical type of horizontal bar chart which illustrates tasks, schedules, milestones and deadlines.

This will manage my time effectively and record my progress, some benefits and limitations of using Gantt charts includes:

Benefits	Limitations
<ul style="list-style-type: none">• Easy to monitor and track progress• Easy to schedule• Visual representation:<ul style="list-style-type: none">◦ Clear◦ Easy to understand• Effective planning tool• Makes the project more feasible	<ul style="list-style-type: none">• Doesn't consider difficulty of tasks• Doesn't consider priorities• Doesn't consider the resources required to complete a task• Complex to set up

Introduction

Task	Resources	1	2
Introduction			
Choosing a project and a name	Word		
Create styles for the document			
Set up a cover page	Word		
Table of Contents	Word		
Created reference section	Word		

Analysis

Task	Resources	3	4	5	6	7
Analysis						
Overview of the project	Excel					
Game loop flowchart	Diagrams.net					
Similar Solution	Word					
Feasibility	Word					
Create a gantt chart	Excel					
Languages, IDEs and Softwares	PowerPoint					
Analyse Stakeholders	Word					
Stakeholder survey	Word Google Forms Excel					

Design

Task	Resources	8	9	10	11	12
Design						
Game decomposition	OneNote					
Character class decomposition	OneNote					
Create flowchart functions	Word OneNote Diagrams.net					
Create a GUI designs	OneNote					
Create modular design diagrams	Diagrams.net					
Write algorithms	Word					
Stakeholder game systems discussion	Word					
Testing systems	Word					

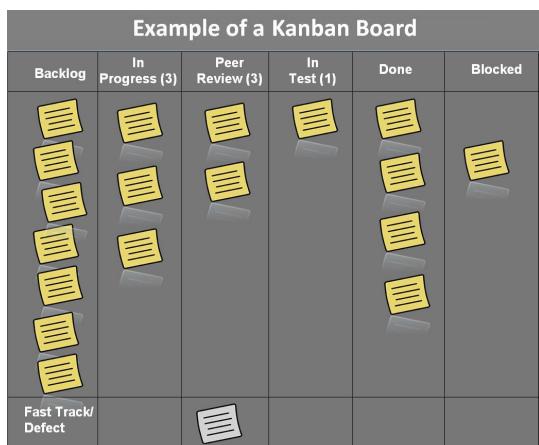
Development

Task	Resources	13	14	15	16	17	18
Development							
Website UI	Replit						
Game Class	Replit						
Map Class	Replit						
Entities	Replit						
Player Movement	Replit						

Task	Resources	19	20	21	22	23	24	25
Development								
Character Transformation	Replit							
Collisions	Replit							
Map Switching	Replit							
NPC Behaviours	Replit							
Battle System	Replit							
Menu	Replit							

By the 25th week, I should have totally completed the project including the documentation, as well as the evaluation section, which is not part of the Gantt chart file at the current moment, because to prevent cluttering the gantt chart and also because it is not entirely necessary to add it because the evaluation section could be completed in matter of few hours.

Overall, I am pleased with the project's estimated timetable since it allows me to properly balance everything. I have overestimated the time it will take me to complete this project since I have not finalised every feature that I intend to add to the game and have just temporarily included the basic components. I will update this gantt chart once I have completed all of the development process components.



I could use a different approach to track my progress and manage my time such as using a project management tool called '**Kanban Boards**'. This tool focuses on continuous improvement, encourages collaboration, and emphasises workflow. However, I am more inclined to stick to Gantt Charts as Kanban Boards are better for team projects, but I am managing this project by myself therefore there isn't any need. Moreover, Kanban Boards will not allow me to track my progress and time, which is really important for this type of

project, especially for the development process because it will allow me to identify areas and features where more resources and time is needed in order to complete them.

Typically, RPGs are games with long-term goals where most players play endlessly. Because of its complexity and the number of features that RPGs typically include, this appears to be a time-consuming project to code. Some features and modules would include: the battle system, maps system, movement system, interaction system, and dialogue system, just to name a few. This might not seem feasible, especially coding these modules in a new language; however, my plan is to code these separate modules in a language that I am already confident in and translate them into this new language.

Languages, IDEs and Softwares

I have decided that I will be coding mostly with Javascript on the Replit IDE alongside other software and programs, such as video and photo editing software, to draw pixel graphics, create and edit game cutscenes, and add more animations to the game.

Javascript & Replit

Using JavaScript and Replit IDE is the best optimum approach for developing a 2D RPG Game.

I chose JavaScript because it offers a wide range of tools for constructing interactive and dynamic web-based games. Because this project will involve complicated mechanics such as processing player input and game events, this programming language enables for simple management of graphical aspects and rapid development. Because it supports a broad selection of libraries, including those particularly built for producing 2D games, Javascript may make development easier and more efficient.

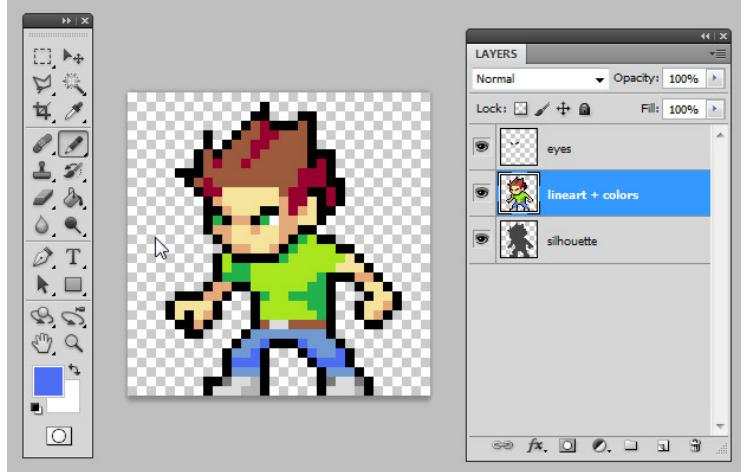
I could use Python which I am more familiar with and I feel like it's more easier for me to use because python syntax is highly readable and easier to understand and it also supports easier syntax and logic. Moreover, Python has numerous libraries, one of which is Pygame, usually used to create 2D games like, which is more reason to believe that Python would be a more suitable option as a programming language over JavaScript. Using Python, however, may lead to some limitations because it does not provide performance optimisation compared to JavaScript. Furthermore, PyGame is not suitable for my game because I don't like the fact of how it handles graphical elements and has several performance problems, and also requires an installation process, whereas javascript games can easily be accessed through the web instead of using an executable file. My original method of building this 2D RPG game in JavaScript may be a better fit because it is meant for web-based games like mine, it has superior graphic capabilities and animation because JavaScript is a client-side scripting language, it requires less hardware, and it is more efficient.

Second, I'm using Replit IDE, because of its high customisation options, interactive and easy to use UI, it also provides more technical support and benefits such as reloading and auto-save features which will definitely speed up development. This IDE also supports Git, facilitating version control, allowing the developer to track their progress and makes iterative testing much easier. I could use Visual Studio code instead, which is more popular, has increased customisation options and includes a powerful code editor. Not only supporting a vast range of useful tools and features, but also allowing developers to debug their code easily. However, Replit is a better IDE for this project because it allows you to easily create and deploy web applications, as my project is going to be a web-based game, further supporting more web-based tools than VS code

Softwares

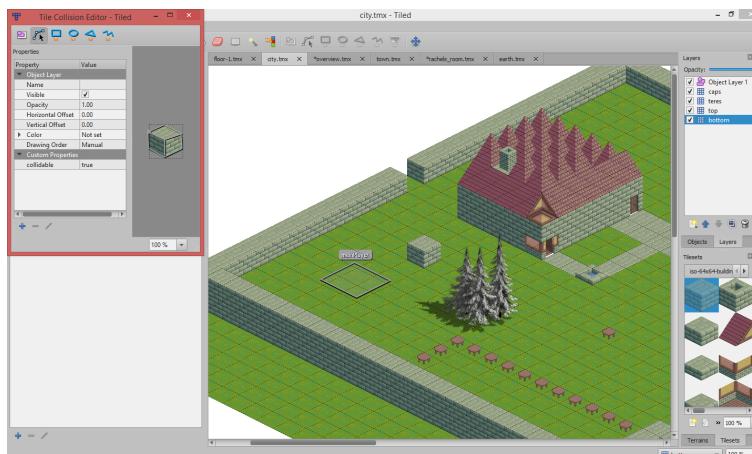
Talking of graphics, I will need to learn how to digitally draw pixelated graphics on a computer, which is a foreign skill for me, as a complete beginner into graphic design.

I will need to watch tutorials on how to make pixelated sprites for video games, learn to animate and use sound effects for my game to add to the complexity.

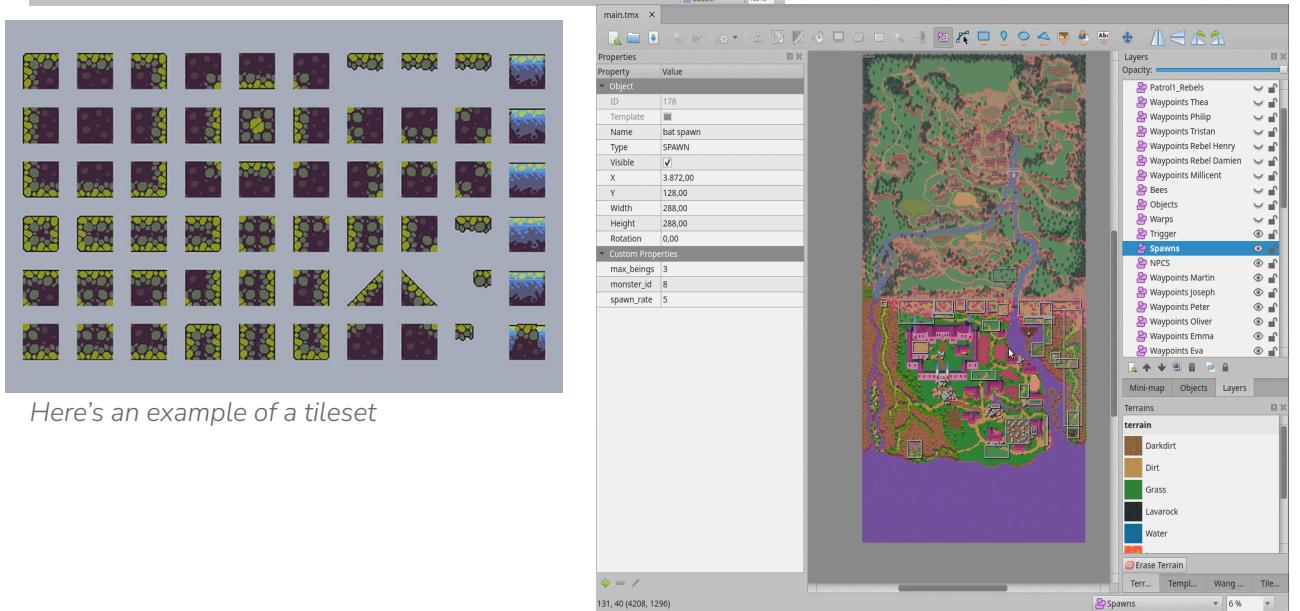


I will be required to try new softwares that will let me produce such features, such as **drawing software, video editing software for animation and sound editor software**.

Example of a pixel drawing software



There's another software that I will be utilising to create maps, which is called '**Tiled**', this allows me to **create maps using tilesets and images**, adding them in an empty canvas, and connecting each tile with one and another to create a map



Here's an example of a tileset

Tilesets are a collection of graphic images which are laid out on a grid to create various types of graphic images such as maps.

Here's an example of a user creating a map using Tiled

Tiled

This software allows you to create as many maps as you like in your own way, it also gives the user more freedom and flexibility when creating maps, the user can use their custom tileset, they can change the size and resolution of the canvas and separate sets of images into layers.

I could use pre-made maps online, which are more high quality as they are created by experienced map creators. Also, creating your own maps is really time consuming and using the software could be difficult for beginners. However, I am already familiar and experienced with the basics of the software to produce a good set of maps using Tiled.

Moreover, using pre-made maps may lead to some limitations to the game, for example, most pre-made maps don't come with JSON file which store their collisions coordinates, whereas if I create my own maps, I will be able to export the map as a JSON file so I cause my collision detection approach in my game appropriately.

Hardware & Software Requirements

Identifying hardware and software requirements are important because it allows the developers, such as myself, to determine what the user requires to be able to play the game and enjoy a smooth gaming experience. However, there aren't many high requirements for my proposed solution because of it being a browser based game, which can be accessible on any browser.

Hardware	Minimum Requirement	Justification
RAM	4 GB RAM	Ensure that the game runs smoothly without any performance inconsistency or issues.
Processor	Intel Core i3, any equivalent AMD processor	Requires any modern processor to ensure the game runs without any lag. Most devices nowadays have processors that are more than sufficient so it shouldn't be a major restricting requirement.
Graphics Card	Integrated graphics card	Requires only an integrated graphics card to be sufficient to run the game without any graphics issue. This allows
Internet Connection	16 mbps download speed OR 2 mbps upload speed	Requires a stable internet connection of at least 16 mbps, not considering other connected devices, to ensure loading game content correctly and to communicate with the game server, allowing for a smooth consistent gameplay experience. The game could allow for a slower internet speed than 16 mbps, however, the user may not fully experience a reliable and enjoyable gaming experience and may create some errors.

My proposed solution will only be playable on computers, laptops, desktops etc. I could allow mobile users to also play the game either using a browser or an app to play on their mobile or tablet devices, however, this will require me to extend my project further and complicate the development process because I will also have worry about the game mechanics for the mobile devices, as they won't have access to keyboards. I haven't mentioned this in the hardware requirement, however, I am going to specify the operating system in the software requirements instead, because users will still be able to play from mobile devices if they emulate any modern computer operating systems.

Since my proposed solution is a browser-based game, there isn't going to be many software requirements other than the browser and the OS;

Software	Minimum Requirement	Justification
Web Browser	Any modern web browser that supports HTML5, CSS and JavaScript	Most computers already have a pre-installed modern browser such as Chrome, Microsoft Edge, Safari etc. which is sufficient to be able to play browser games. I could make the browser into a executable instead of a web browser, which may lead to improved performance as the game has direct access to memory and use more hardware resources for smoother gameplay, however, this would make the game less accessible as web based games can be accessed by using a direct link, whereas the executable game will require installation which may be time-consuming for the user.
Operating System	Windows, MacOS, ChromeOS	Requires any modern operating system. I could not require an operating system to allow accessibility to more users, such as android or IOS users. However, it would mean extra coding and the lengthening development process, as I would need to add buttons and other user interactions for mobile users.

Limitations

However, there could be some technical limitations of the game due to lack of hardware requirements:

One of the technical limitations that I may encounter throughout the development of this project could have a negative impact on the quality of the game. Since I am not using a fancy language or a game engine for my proposed solution such as unity or c#, it could restrict the capabilities of my game because of the lack of complex physics and graphics.

Moreover, javascript does not support proper memory management and has limited access to low-level hardware, unlike C++ or Java, languages which are more suitable for game development, which can be considered another technical limitation to my project because it restricts improved performance as the game does not have direct access to graphics card or memory. I could use a game engine and more suitable game development programming languages to provide improved performance and complex physics, however, there are several reasons not to because I want to keep my game simplistic, which is a good reason to avoid more complex graphics and physics. I also don't want my game to access the hardware memory and graphics card, this will mean that users with low-end devices may not be able to enjoy it if the game runs entirely on the power of their devices. This is an obvious issue and contradicts the purpose of my project, which is to let every user, no matter their device's performance, enjoy a smooth game experience.

Another technical limitation that the game may fall into is inconsistent performance, even though most users will experience the same game experience and performance as the game does not use any hardware, my game will still be a browser-based game, only accessible through a Wi-Fi connection. As a result, my game heavily relies on web technologies such as Javascript, HTML and CSS, which not only limits complex graphics as I mentioned previously, but also the game's performance will rely on the user's internet connection to provide a decent experience. This may limit performance for users with a weak Wi-Fi connection leading to inequality in performance of this game between different users. To fix this issue, I could use executable games, which does not rely on the user's Wi-Fi connection and it can be accessed without any internet connection, however, this could create more limitations for the future development of this game, such as not being able to add a multiplayer system, not being to allow for easy updates and restrict compatibility with other devices. Whereas browser-based games allow for easier accessibility and distribution without the need for a complex installation process, increased compatibility as most devices have browsers, allows for faster and easier automatic updates and have lower system requirements than executable games.

Success Criteria & Requirements

Note: I have integrated the **requirements** from my stakeholders and the **success criteria** into one table because it provides a straight-forward way to confirm that the project achieves its intended goals and a clear and quantifiable technique to assess whether the requirements have been effectively met.

Success Criteria	Evidence	Justification
Multiple different maps	Screenshots of different maps	The first criteria is for the game to include many different maps in order to create an engaging world to explore for the user, as exploration is the main purpose and goal of my proposed solution. Instead, I could utilise a single map, which would be easier to manage and work with, decreasing development time. Adding numerous alternative maps, on the other hand, is a preferable solution because it allows for a more diverse gameplay experience, supplementing the game with a stronger sense of discovery.
Character Movement	Video of the character moving with the right direction input from the user.	The second criteria is to allow the user to have a smooth and responsive character movement, ensuring that the game is enjoyable for the user. I could restrict the character movement allowing only to travel in two directions or less, however, adding a 4 way character movement system makes my game more immersive, dynamic and even more enjoyable.
Speed Boost	Video of the user toggling on speed boost allowing the character to travel at a faster speed	The third criteria ensures that the user is able to toggle speed boost abilities, which allows the user to perform more actions, faster travel, and an increased sense of control for the user over their character's movement. I could not add this speed boost feature to keep my game simplistic and reduce game mechanics complexity. However, adding the speed boost feature will add an additional layer of gameplay depth and allows for more strategic movement options.

Character Walking Animation	Video of the character running through a walking animation with the correct frames.	The fourth criteria emphasises on character animation movement which aims to provide players a sense of realism and immersion. I could not add the animations which will allow me to focus on other features instead, as often the animation systems require a long time to develop because of their complexity. However, adding the animation for the character movement because it will add more visual appeal and enhance the user experience, whereas not adding it will make the movement system and the overall game feel dull.
Character Camera	Video of the game where the map moves following the character's movement	The fifth criteria is the character camera, which is an essential feature that must be developed in order to provide players with a clear view of their characters and the map. This feature should follow the character around, always keeping the character in the centre of the canvas. I could instead add a fixed camera system to simplify the game mechanics. However, the dynamic camera is far better because it adds to the immersive nature of the game.
Special Ability	A video of the character using their special ability.	The sixth criteria ensures that the game must have a special ability feature to add an additional layer of gameplay depth and increase the complexity as it allows the character to perform more actions. I could not have a special ability feature to simplify game mechanics and increase game development time to focus on different features. However, a special ability feature allows for more strategic options to the gameplay.

Effects & HUD	A video of the character using their special ability which will show an effect animation and display a HUD.	The seventh criteria ensure that the game must have effects and HUD to provide important information or feedback during the game experience. I could not add effects and HUD to reduce visual clutter and maintain immersion. However, HUDs will allow the user to absorb essential information about the current state of the game, reducing confusion, and effects will make the game more enjoyable and engaging.
Character Collisions	Video of the character not being able to barge through walls, trees, entities and etc,	The eight criteria sets proper boundaries for the character to travel within. I could not add collisions to simplify game mechanics, however, this may lead to some limitations in developing other features such as map switching or entity interactions.
Customisable player skin	A video of the user changing the character skin.	The ninth criteria must ensure that the game is able to provide the user with customisable player skins to allow players to personalise their character. I could remove this particular feature to maintain the simplicity of this game, however, adding the feature will enhance player satisfaction as they are able to choose what they want to play as.
Map Switching	Video of the player exploring and switching between different maps while displaying a transition.	The tenth criteria ensures the game must have a map switching mechanic to allow players to discover and explore new maps. I could not add this feature as it may make development more confusing, however, if there are no map switching features, there is no point of having multiple different maps as the user cannot have a system to access and travel to them, restricting exploration.
Entities in each map	Screenshots of entities for every map	The eleventh criteria ensures the game is able to handle different NPCs and monsters present on the different maps, making the game feel more dynamic. I could only add one or two entities, as many entities may clutter the maps, however, a variety of entities would make the game more engaging.

NPC behaviours and collisions	A video of the NPCs moving in specific patterns and colliding with the player and other entities.	The twelfth criteria ensures that the game will have NPCs with unique behaviours and collisions between other entities and the player, to make the game feel more alive. I could use a NPC behaviour template in order to save effort and reduce the length of the code in the developing process, however, this will make every NPCs behave the same which the user may find boring. Including more authenticity for each NPC will provide a more immersive experience
NPC Interactions & Dialogues	Video of an entity responding to a character interaction with a dialogue	The thirteenth criteria will give NPCs a purpose to be included in the development of my game by allowing the player to interact with NPCs and display important information. I could make the NPCs static and do nothing but move around, however this may have some drawbacks because NPC interactions can allow the player to initiate the battle state. Aside from that, incorporating NPC talks would make the game feel more interactive and will convey more information to the user, making the game feel more interesting.
A battle system	A video of a turn-based battle system where the player is able to attack the opponent	The fourteenth criteria will allow the player to battle other entities or monsters in a turn-based battle system, to increase the difficulty of the game and provide challenges. I could skip the battle system which would save a lot of time as it is one of the most lengthy component to develop in this project. However, adding a battling feature could give the user a sense of accomplishment and progression.
Menu	A screenshot of a menu window with different options and settings for the user to configure with	The fifteenth criteria ensures that the game must have a menu for the player to access different features of the game such as customising character sprites etc. I could not include this feature to increase immersion, however, not having a user-friendly and easy to navigate menu will restrict the user from accessing other features which is quite important.

DESIGN

Objectives

There are many options and features to choose from but not every feature and function can be added to the game. To figure out the best option to add we need to investigate what our consumers will like between the multiple features.

To fetch their inputs we can again use google forms and send the link with their preferred method of communication. I have used google forms like before because of its extensive compatibility and accessibility.

Game Design survey

What kind of menu would you want?

1. Fullscreen: 2
- 2. Centre Peek: 3**
3. Side Peek: 1

What would you want the game canvas to be?

1. Fullscreen: 2
2. Custom: 2
3. Centre rectangle: 2

Should I include music? If yes, what genre?

- 1. No: 3**
2. Ambient Music: 1
3. Retro Music: 2

What kind of overworld?

1. 5+ Small Maps: 2
- 2. Max 4 Large Maps: 4**

What kind of exploration should it be?

- 1. Open world: 4**
2. Progress with story: 2

What kind of transition would you like?

1. White screen fade: 2
- 2. Black screen fade: 4**

What should the interaction/proceed button be?

1. Space: 2
- 2. Enter: 4**

Should the player be able to rematch enemies for grinding levels?

1. Yes: 1
- 2. No: 5**

Should the NPC dialogue include a face set of their sprite?

- 3. Yes: 6**
4. No: 0

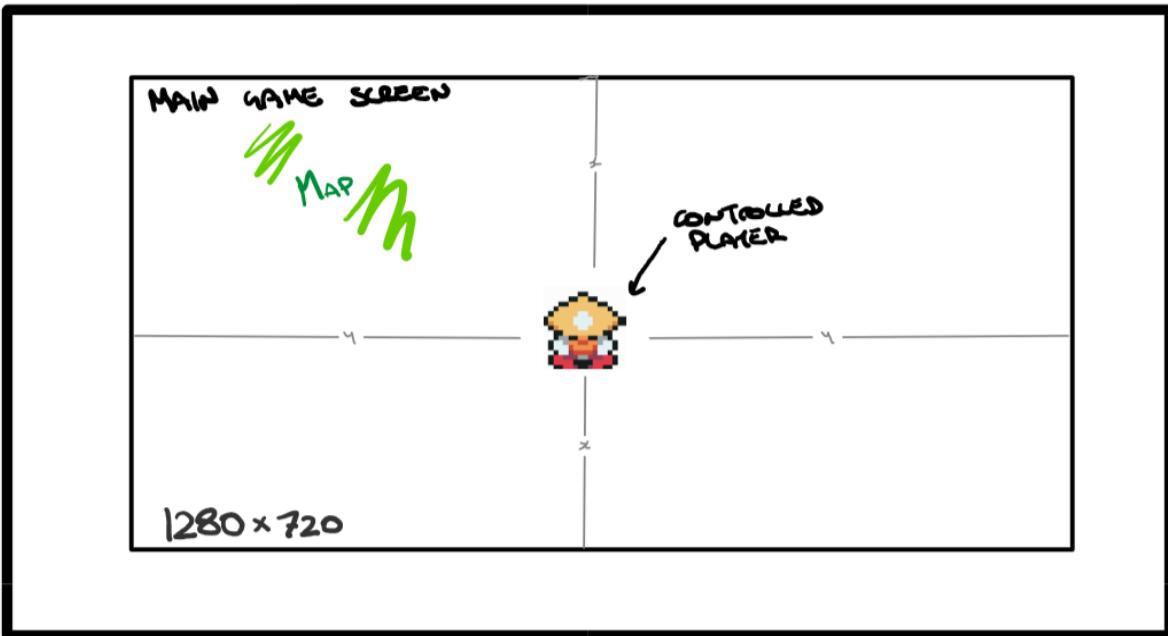
This information will help me analyse which features I will most likely add to the game by only choosing to add the feature voted by the majority, as the feature with the most votes is what players will want, as a result of these properties being added, it will increase the overall enjoyment and fulfilment of the player.

GUI

Because they offer an intuitive and interactive means to develop, alter, and test various game elements, such as levels, characters, items, and environments, GUIs are crucial for game creation. Without having to deal with complicated coding or technical intricacies, game developers may quickly see and modify various game elements with the aid of a well-designed GUI, as a result, this can easily speed up the development process.

I could not plan and design any GUI and just develop as I go by, because creating and analysing GUI is time-consuming and challenging for me as I have to deal with limited resources or time constraints. However, despite the challenges, it is better to design a GUI to enhance the overall user experience and make the game more accessible to a wider audience.

Layout GUI

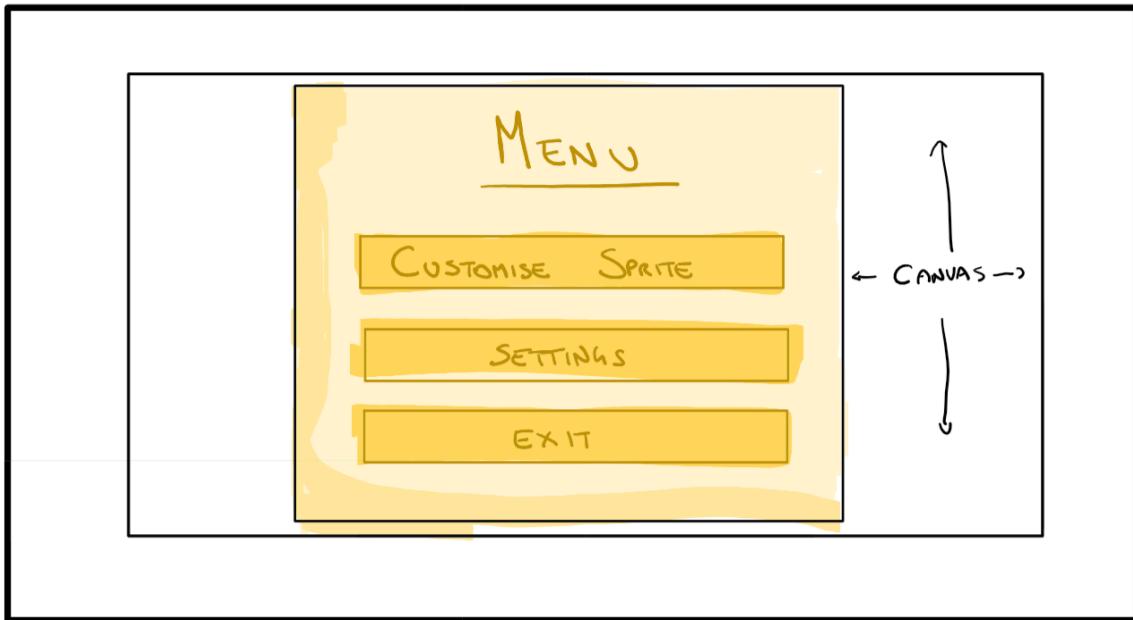


have drawn the layout of the browser-based website which will show the game canvas in the centre and the HTML and CSS structure of the game. The actual game canvas will be in 1280x720 pixel resolution.

I could have made the canvas fullscreen to create more immersion in the game, as it makes the user feel more engaged as it removes unnecessary empty sections. However, adding a fullscreen game canvas would make my game incompatible with other devices and browsers, furthermore, I will need to configure the game canvas to fit in every device, and increased size of the canvas will decrease the quality of the game because of the size of the sprite sheets and graphics that I am using requires a specific dimension, in this case a 1280x720 resolution.

Additionally, as seen in the GUI diagram from the user's perspective, the character would be placed in the centre of the game canvas, when the player moves their character, the game should follow the character to always maintain their position in the centre relative to the canvas, I would be using the character camera feature to successfully execute this feature. I could instead add a fixed camera system to simplify the game mechanics. However, the dynamic camera is far better because it adds to the immersive nature of the game because it provides a clear view to the user so they have a sense of direction and where they are in the map.

Menu GUI



By pressing the ESC key, the user can toggle the menu GUI. This GUI contains a menu window onto the canvas, which shows the different options available for the user to choose from such as the customised sprite option, which will allow the user to change their sprite design, in order to meet the success criteria of customizability of character sprites. The second button is the settings option, which will allow the user to change the audio, brightness etc. and change their preferences in the game. Finally, the exit button allows the user to quit the game whenever they choose by allowing their browser to go back one page. I could have taken a different alternative approach such as not adding a menu to create a sense of immersion, reduce the amount of executable actions in the game, which may be a sensible approach to reduce the clutter of different mechanics and to simplify the overall game. However, this could restrict and create limitations to access other important features of the game such as the character customisation option. This specific feature will allow player to choose any sprite entity as they wish and play with it, by committing the new sprite to the player instance

Overall, I believe that the menu GUI looks appropriate for the user to access different features of the game, allowing them to perform more actions, and the buttons are big enough for the user to click and read the instructions of each button.

Dialogue GUI



The dialogue GUI is only accessible if the user interacts with the NPC, which it then outputs the NPC's dialogue, NPC's face set and the NPC's name, as shown in the diagram above. I have positioned these elements on the bottom of the canvas, the user can clearly see these elements because of its extensive size as it stretches from the start of the canvas from the left, all the way to the end of the screen to the right of the canvas, and vertically covering $\frac{1}{3}$ of the canvas. The faceset of the respective NPC is positioned on the left side of the dialogue to give the NPCs more personality. The name of the NPC is positioned on top of the faceset.

Overall, I believe that the position of these elements is best positioned at the bottom as it is a common approach in most existing solutions. I could have positioned these elements somewhere else to differentiate from most other games and to be more somewhat unique, giving more personality to my game. However, positioning it at the bottom will allow the user to feel more familiarity with this game and increase the consistency of the game.

The font of the text in the dialogue feature of this game is going to be really important and I will be required to consult some of my stakeholders to assist me in choosing an appropriate font which will appeal to my larger audience of players.

I could create a survey and ask for feedback or provide sample fonts to every single stakeholder, however, this approach will not be suitable because not everyone will have experience with font selection and design, which will lead to incorrect feedback and a misrepresentation of what the audience will want. Therefore, I am only going to ask for feedback from two different stakeholders who I believe are suitable with this task and will allow me to customise and specify the questions only for two stakeholders rather than the rest.

Stakeholder consultation

Q: What are some significant factors a game developer should consider when selecting a custom typeface for in-game text, such as dialogue and UI elements?

Bartosz: "The way a typeface mixes into the game's general style is, in my opinion, one of the most crucial things to consider when picking a distinctive font for in-game text. Choosing a pixelated font will surely assist to improve the 2D pixel art theme of your game and give players a unified visual experience. Even with a pixelated font, legibility and readability must be considered to ensure that the text is easy to read and comprehend."

Q: What do you believe are some errors to avoid when it comes to choosing font choices and accessibility with your experience with customer support?

Ivo Igor: "In my previous experiences I have encountered many games and websites where the font choice did not match the standard look of the game or made the text more challenging to see, which could cause confusion and annoyance from the user's perspective. This is obviously a major issue with accessibility as some people won't be able to read the information provided in the game, which is really unappealing."

Custom Font



After consulting with the stakeholders, I have concluded that I am using this font, which follows a pixelated theme that will fit well with the overall design of the game and appeal to the users. I have ensured it met the requirements from the stakeholders and they have approved this font to be used in my game.

Battle GUI



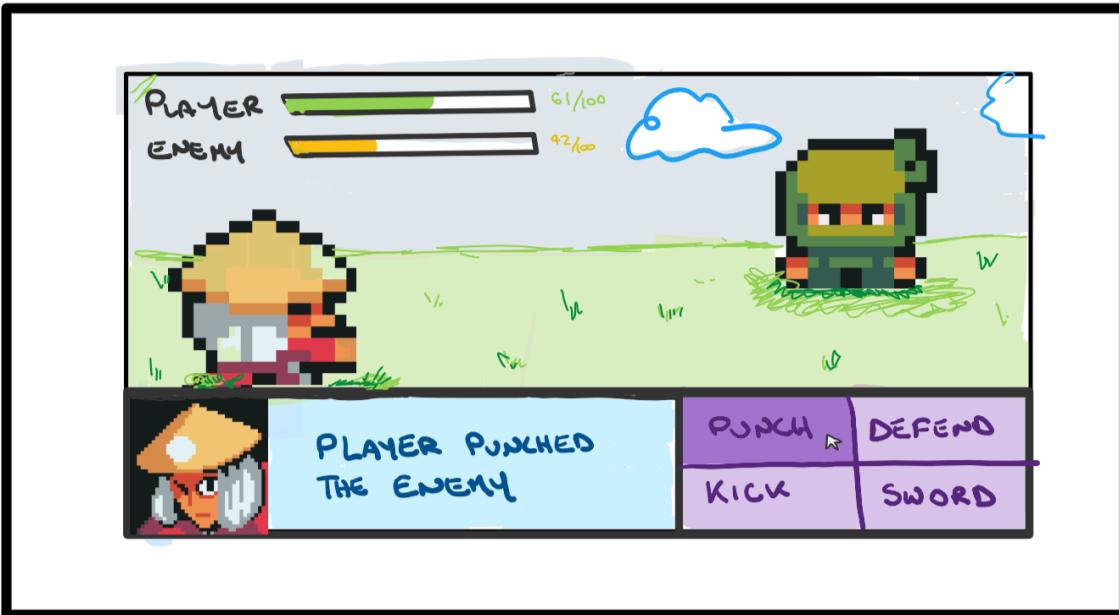
The battle GUI will have a different state from the overworld, where the player is able to attack an enemy and be given several fighting options to attack the enemy. This feature will see the game switch between different states, such as the overworld and the game when the player enables this feature by interacting with an NPC or monster.

I have added the health bar on the top-right of the new game canvas, which will use a HUD class and algorithms to handle the health bar status etc.

The position of the health bar being on the top-right is a common approach in many RPG games with turn-based battle systems. I could position it closer to the battle dialogue and fight options, so the player won't need to focus on different areas of the canvas, but instead keep their eyes on the bottom of the screen. However, this alternative approach may lead to clutter and may make it difficult to take information from the canvas, as everything is in one place, making it more unappealing to the user. Therefore, positioning the health bar separately and further away from the battle dialogue is a better approach, also because there is more empty space on the top.

Furthermore, I believe that the position and the size of the game dialogue and the fight options is better to be placed on the bottom of the canvas, which is the same structure used in the Pokemon battle states, allowing the user to absorb information about the status of the battle and allowing them to perform certain actions on the fight option box.

The only interactable button on the Battle UI is the fight option, when click the game will show a variety of 4 options as shown in the following UI diagram:



The box on the bottom right hand side corner seen in the GUI is called a battle option box, which when it is clicked, the battle box will display some actions that can be performed during the battle state as shown in the GUI above. When the player hovers over an action, the colour of the button should change to a darker shade, this hover feature is really common mostly in websites and only requires one or two lines of CSS, therefore, it doesn't require much development. Then when the user clicks one of the 4 actions, the battle will show that action was executed by changing the player's sprite to 'attack mode'. After any attack move, the enemy's health should deplete. After that process, the battle dialogue will be updated to show what action has been performed, by who has it been performed by displaying the name of the user and the face set of their character, the health bar will also be updated to show each entities' current health. This feature will obviously require a ton load of development because of its complexity and numerous sub features required to develop the whole of the battle system.

I could have pursued a different approach which is really common in most RPG games with a turn-based battle system which is to allow the user to select their desired actions by navigating to it using arrow keys. Those retro games, however, were only accessible using keys which could only support arrow keys to allow the user to select their options, most of these franchises have moved on to support mouse cursor or touchscreen, and since my game is a browser based game. Therefore, allowing the user to input using touchscreen or cursor for this battling system is far more efficient because they already have necessary equipment.

Software Design

Software design is the process of designing software systems which includes implementing code using appropriate design principles and techniques.

Decomposition

Decomposition is the process of breaking down a complex problem into smaller problems which can be solved more easily. This process can be depicted in multiple methods, but I will be using hierarchy diagrams to decompose my game, which will allow me to break it down into individual smaller concise classes to visualise the problem better. I would instantly know what tasks I need to complete in order to progress and finish working on the whole problem. I could just use brute force, trial and error, which are more time-consuming and inefficient compared to decomposition, but it obviously requires less planning. However, using hierarchy diagrams will allow me to properly visualise the problem, allowing me to establish the necessary measures to follow to address the problem. Furthermore, my method gives a methodical method for breaking down a problem into manageable components.

I will be using **Hierarchy Charts** to decompose the problem. This approach displays the structure of the problem by illustrating links between classes and subclasses and how they fit together to solve the problem.

Object Oriented Programming

Object Oriented Programming, often referred to as OOP, is a brilliant paradigm for decomposition, which is a programming model which revolves around using objects rather than logic and functions.

Using OOP will allow me to structure a whole problem into simple reusable components which are the blueprints to create objects, these blueprints are known as “class”. A class stores attributes and methods, which can be configured and reused an infinite number of times to create objects, also known as “instances”.

OOP makes structuring a problem easier and works well with decomposition as each individual problem can have its own class, these classes are called “subclasses”. All these smaller classes make up the whole solution.

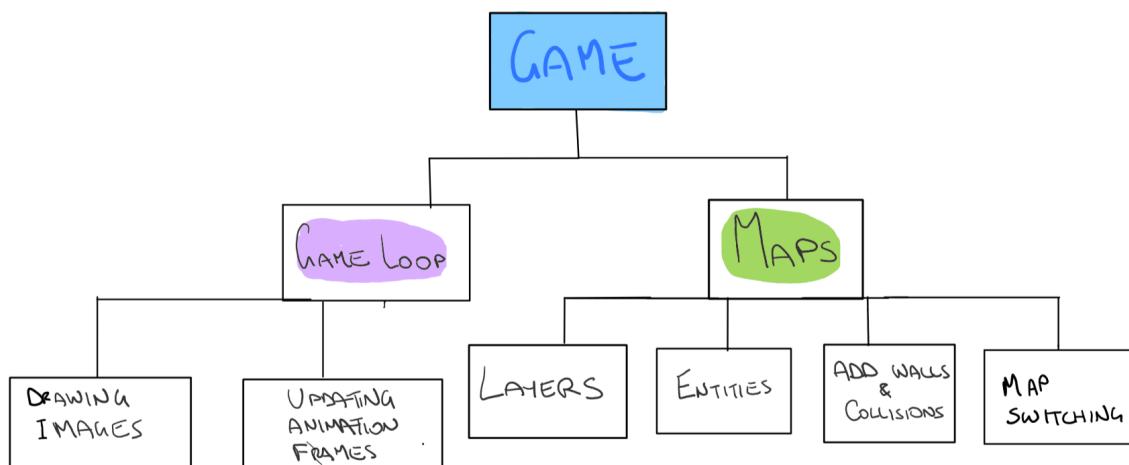
I could use other programming paradigms such as procedural programming, which is easier and straightforward, however, OOP is more effective and less time consuming as it allows easier debugging, provides better flexibility and allows extensive reusability.

Now that I have explained the software designs I am going to use, I can start planning the design of the game which will help me ensure the final solution is well organised, functional and meets the desired objectives and requirements.

I could not plan the design, which will see me skipping the design section and heading straight to development, however, this alternative ‘lazy’ approach lacks clarity, as it may cause some confusion about how the game should look like, also resulting in a disorganised final solution. Moreover, it will lead to increased risk of errors, inefficient development, as the code will take longer than necessary, and not planning the design of the game will most definitely lead to a poor user experience as the game may not be properly designed, miss some essential elements and features and may cause user frustration

There are 3 main components of this proposed solution, the most important being the main class system of the game which will link and ‘create bridges’ for every other subclass and components of the game;

Game Class



I have used a hierarchy chart as mentioned for the game class. This class will store and initialise the game loop, which will make sure to draw images constantly, update the game’s state and update animation frames by iterating the drawing functions over and over again. The game will store the map instance as an attribute which will allow the game to connect different subclasses such as the player or entity class to the map class. This approach will facilitate and allow for the development of many essential components in the game such as map switching, drawing entities, (as the map instance stores a dictionary of entities that it's associated with that map) and also collision detection, all because the player will now be able to interact with the attributes and other functions of the map instance. To understand this better and plan it more effectively, I will need to plan the attributes and methods of each class, following the Object-Oriented Programming construct.

This will complement the hierarchy charts and I will have a much clearer understanding and approach to developing the game class during the development process.

Game Class
Attributes:
<ul style="list-style-type: none"> ● Canvas ● Map
Methods
<ul style="list-style-type: none"> ● Loop() <ul style="list-style-type: none"> ○ DrawImages() ○ Map Switching() ● Init()

The **game class** will contain the canvas attribute, where the images will be drawn, and it will also store the map attribute, which is the current map instance that the player is on right now. The map attribute will allow the game loop, which is the Loop(), to know which images to draw, the images mainly depend on the map instance as it stores the images for the map layers, entity images and position and so on. When the map is exited due to the player travelling to a different map, the game loop will make sure to switch the maps and update the game's state.

To initialise the game loop, an init() method is required so that the game loop is running continuously, the init() method will mainly configure the initial variables for the game such as setting the map, fetching the coordinates, creating the player object etc.

The map class is also one of the main components of this game. I could have integrated the map class into the game class instead of creating a class, however, this could lead to inefficient use of code, harder to read and unnecessary lines of code in one file. Creating a new file to store a different class will be a better approach, as it increases the modularity of my project.

The map class will store the layers, the entity objects and the collision coordinates stored as walls. The methods include drawing the layers of the map, checking the collisions by first fetching the collisions between the player and the map or the entity. The map class also has an update function where it updates the state of the map such as the player's position after map switching.

Map Class
Attributes:
<ul style="list-style-type: none"> ● Layers ● Entities ● Walls
Methods
<ul style="list-style-type: none"> ● DrawLayers() ● CheckCollision() <ul style="list-style-type: none"> ○ MapCollision() ○ EntityCollision() ● FetchCollision() ● Update()



Lower Layer



Game Objects



Top Layer

Game Loop Algorithm

```

PROCEDURE Loop():
    // Define the game loop function
    FUNCTION gameLoop():
        // Update game state
        // ...
        // Request the next frame of the loop
        requestAnimationFrame(gameLoop)

        // Initiate the game loop
        CALL gameLoop()
    END PROCEDURE

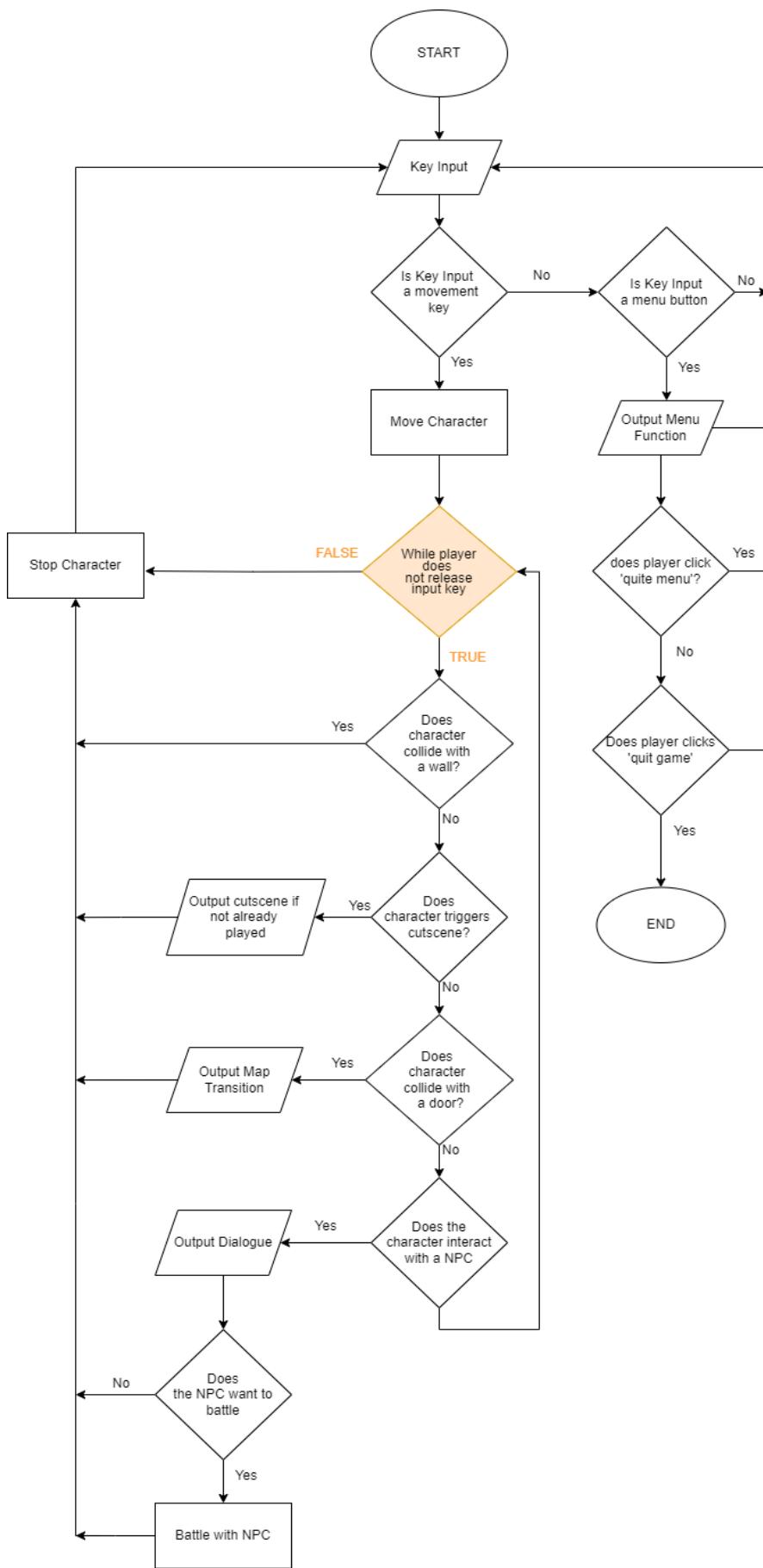
```

The algorithm above forms a complete solution to the problem of rendering, drawing and updating a game at a consistent frame rate.

The algorithm uses a **requestAnimationFrame()** function which requests the next frame of the loop. This computational method optimises the animation and will execute a set of instructions before repainting the canvas. This allows the game to run at a consistent rate, regardless of the user's computer speed or other factors that could affect the frame rate. Therefore, this solves the initial problem where users are unable to play certain games because of their hardware requirements, this algorithm fixes this issue. The **Loop()** function will be the method of the game class, where this method will be initially called using a **init()** function, the purpose of the **init()** function is to start off the game loop after loading all the files and setting the initial values for global variables that is needed for running the game. This **Loop()** declares a new variable **gameLoop()**, which runs the specified instructions within itself, such as drawing the images, rendering graphics, and mainly updating the game's state. This ensures that the game is updated at a consistent rate. Overall, this algorithm provides an immersive gaming experience for the user due to the consistent frame rate that it achieves using various methods.

I could use a different approach such as using 'setInterval()' methods to update and render the game at a consistent rate. This approach will make the development process much easier and is more efficient as it shortens the length of the code needed in the overall game. However, the main flaw of this alternative approach is that the 'setInterval()' method does not account for browser performance and may run at a slower or faster rate depending on the browser's workload, leading to inconsistent frame rates and does the opposite of our problem; where we want all users to have the same experience regardless of the hardware they use. Moreover, this algorithm will aid to develop the complete system for the game as everything will revolve around this algorithm, every update and drawing function will be called within this game loop algorithm, in simpler terms, this algorithm is the heart of the game.

Game Loop Flowchart



This figure shows the overview and how I am going to structure the code and the flow of the game.

The game will include subprograms and functions such as the player interaction, game menu, collisions etc. which are not fairly detailed in this figure as these functions will have separate diagrams and flowchart.

I will be using this flowchart diagram to help me implement my game and most traditional 2D RPG games follow the game loop structure, where the only way a player can exit is through by toggling the menu screen, and generally the player can't die and end the game which is a common trend in other non-RPG games.

Collision Detection Algorithm

The collision detection algorithm will be stored in a different class called Map, which will also draw the map layers, store entities and support map switching as seen previously in the hierarchy chart of the game. The collision detection will use a json file with data which provides information about the walls of the map. I will require two algorithms. One algorithm will traverse the map.json file collision data, which is stored as a JSON object, this will allow me to store the coordinates of collisions in an attribute of the Map class.

Here's an example of a JSON file collision data for one of the maps:

```
{
  "layers": [
    {
      "data": [0, 0, 0, 375, 637, 638, 638, 639, 378, 0, 0, 1, 379, 0,
              0, 375, 600, 398, 0, 0, 0, 0, 491, 619, 620, 617, 389, 0,
              375, 597, 616, 0, 0, 0, 0, 0, 0, 0, 0, 397, 379,
              0, 613, 0, 0, 0, 629, 647, 648, 649, 0, 0, 0, 0, 0, 0,
              385, 631, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 389],
    }
  ]
}
```

0 represents free space, where the player can travel to. Whereas anything else other than 0, is a collision block where the player cannot travel to. I can easily generate these collision JSON data by exporting my maps in a JSON format when making them on Tiled.

```
METHOD fetchCoordinates(){
  fetch('Map.json')
  .then(response => response.json())
  .then(json => {
    this.walls = json.layers[0]
  })
}
END METHOD
```

The fetchCoordinate() method fetches the collision data in the JSON file of the current map instance and stores it in the this.walls attribute of the map class. Now we have the collision data of the map.

The other algorithm will make sure that the player is not able to travel through a coordinate which is in the

```

METHOD checkCollision()
    // Check if the player is within the map boundaries
    IF player.x < map.width AND player.y < map.height THEN
        // Loop through each wall object in this.walls
        FOR EACH key IN this.walls DO
            // Check if the player is colliding with the current wall
            IF this.walls[key].data[player.x][player.y] EQUALS 0 THEN
                RETURN "free space"
            ELSE
                RETURN "collision!"
            END IF
        END FOR
    END IF
END METHOD

```

This function checks for any collision between the player and the walls of the map instance. It starts off by checking if the player is within the bounds by comparing the player's coordinates to the width and height of the map instance using an if statement.

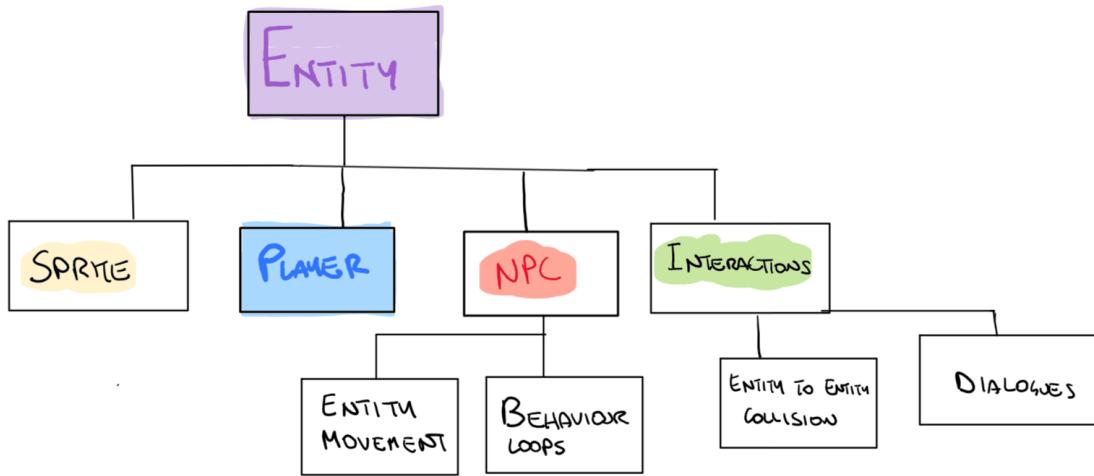
If the player is within the bounds, the function will then proceed to iterate through each wall object by taking in the player's coordinates.

The `this.walls[key].data[player.x][player.y]` represents the data from the json file, where the element is either 0 (free space) or anything else (collision). If the player's coordinates are on free block, meaning that it equals 0, then there is no collision, Otherwise, the code will return a collision. This algorithm is collision detection, where it only detects collisions. I can now use this algorithm to add working detection by adding more logic to the player's class during the development process.

Overall, this algorithm solves our problem as it provides a working collision detection that we can use and implement into our game to add boundaries to where the player can travel.

I could use a physics engine for implementing a grid-based collision detection system, such as Matter.js or Box2D, which provide accurate collision detection which detects any collision between entities and objects within the neighbouring cell, this also could be more efficient for checking collisions. However, using a physics engine for a simplistic game such as mine would be overkill, introduce unnecessary complexity and require more resources to implement this system. Furthermore, my algorithm will also allow us to add more essential features such as map switching and player interactions, making the game more immersive for the user.

Entity Class



The entity class will be one of the main components of this game, thus why it's one of the few classes that requires some form of decomposition in order to develop the entity class by minimising errors and providing me with a clear approach.

I will also require a sprite class, which focuses on the images and animation of the entity that will be required in order to fully develop the entity class.

I have created a player class as seen in the hierarchy chart, where the player class inherits from the entity class, as the player is considered an entity and requires most of the attributes from the entity class plus some additional features which require extra attributes and methods such as the speedBoost feature, the special ability feature and handling of HUDs and effects.

Using inheritance will drastically reduce the length of the code because of its reusability aspects, I could not use inheritance and rewrite a new class from scratch which will give me more flexibility in developing the player class and make things less complicated, however, I will be required to rewrite the same methods and attributes which will make my game less efficient negatively impact performance and lengthen my code by a great margin.

The entity class will let me create NPCs for the game using the template of attributes and methods, the NPC entities will heavily emphasise on the behaviourLoops which will allow NPCs to move in patterns, allowing the game to feel more alive. Also, as I have previously stated, the NPCs will include the dialogue feature, which adds an additional degree of engagement to the game.

Entity Class	Attributes
<p>Attributes:</p> <ul style="list-style-type: none"> • Name • Position • SpriteInstance <ul style="list-style-type: none"> ◦ SpriteSheet ◦ Animation ◦ CurrentFrame • TilesLeft • Direction • Speed • Dialogue • Behaviour <p>Methods</p> <ul style="list-style-type: none"> • Update() <ul style="list-style-type: none"> ◦ UpdatePosition() ◦ UpdateSprite() • BehaviourLoop() • Interact() <ul style="list-style-type: none"> ◦ DrawDialogue() 	<p>As shown in the diagram, the entity class will have numerous properties, including the entity's name, the position of the entity on the map, and the spriteInstance, which is the image of the entity being drawn. These are the minimal basic attributes required to draw the entity instance.</p> <p>The entity class will also contain a set of attributes for their movement, the TilesLeft attribute stores the distance remaining to travel, the direction attribute ensures that the entity will face or move towards, and finally the speed attribute determines how quickly the entity is able to travel. The speed attribute in this game is most likely going to be toggled on and off to accommodate the speed boost feature. Furthermore, the dialogue and behaviour attribute are only going to be used by NPCs and monsters, entities such as players won't need these attributes, the dialogue attribute stores the text the NPC is going to output when the player interacts with this entity instance. The behaviour attribute will ensure that the NPC or monster will have a set of behaviours, where each behaviour will store the direction, speed and tilesLeft. This allows the development of behaviourLoops for NPCs and monsters.</p>

Methods

The entity class will also contain several methods to ensure that the criterias are met efficiently, the main method of this class is the **update()** method which could be called in every game loop iteration, this function will contain other functions such as calling the methods from the sprite instance to draw the entity, alongside calling the **updatePosition()**, which will allow the entity to travel, and **updateSprite()**, which will ensure the movement animation and apply the current sprite to their corresponding direction. I will also add a **BehaviourLoop()** function which will activate the NPCs or monster's behaviours. Finally, the **interact()** method will ensure that the NPC is able to interact with the player and output their dialogue by calling the **drawDialogue()** function in order to respond, where the player tries to interact with this entity instance.

Sprite class

Game Class
<p>Attributes:</p> <ul style="list-style-type: none"> • obj • skin • animationsMap • animationSet • currentFrame • framesLeft • framesLimit <p>Methods</p> <ul style="list-style-type: none"> • updateSpriteSet(direction) • updateFramesLeft() • drawObj()

Attributes

The sprite class will include the **Obj** attribute, which is just an instance of the entity class. I need this attribute so the sprite class can access important data about the entity such as the source image, direction attribute etc.

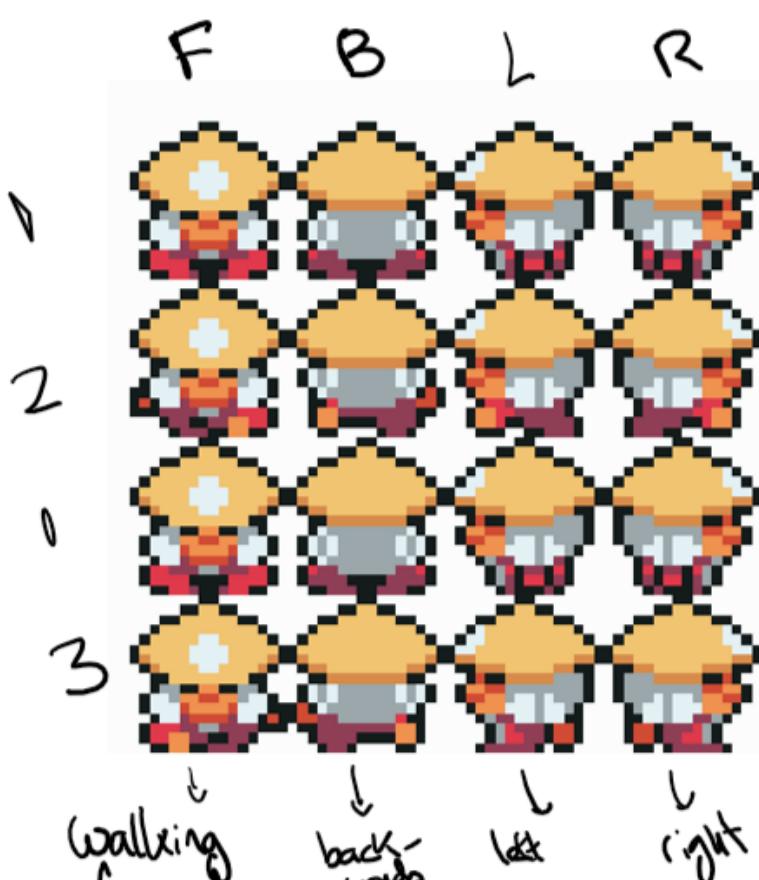
The **skin** attribute will store the spritesheet of the entity. The **animationsMap** attribute will store a map which contains a set of animation sets that the entity can iterate through. The **animationSet** is the current animation that the entity is currently on containing a set of frames to generate the animation when drawn and iterated at fast speeds.

The **currentFrame** is the current display image of the entity taken from the spritesheet using a set of coordinates. The **framesLeft** indicate how many frames are remaining to finish the animation and finally the **framesLimit** holds the limit of how many frames can be iterated in one gameloop iteration, this attribute can be configured to change the speed of the animation, especially useful for the speed toggle feature, where the walking animation would be faster than normal when the user toggles on the speed boost.

Methods

There are only a few methods for the sprite class, one of the methods is the **updateSpriteSet()** method which takes in a direction parameter and updates the animation set of the entity. This function is called when the user moves their character, so the game then can make sure the player's sprite matches the corresponding direction. The **updateFramesLeft()** method will ensure that the animation is iterated through and displayed properly by subtracting the **framesLeft** attribute and updating the **currentFrame** to provide an illusion of motion. The **drawObj()** draws the sprite onto the canvas.

Animation Movement Algorithm



Direction

- "Idle-down" : [[0,0]],
- "idle-up" : [[1,0]],
- "Idle-left" : [[2,0]],
- "Idle-right" : [[3,0]],

Movement

- "Walk-down" : [[0,1], [0,0], [0,3], [0,0]],
- "Walk-up" : [[1,1], [1,0], [1,3], [1,0]],
- "Walk-left" : [[2,1], [2,0], [2,3], [2,0]],
- "Walk-right" : [[3,1], [3,0], [3,3], [3,0]],

This figure shows the spritesheet of an entity which includes the direction and walking animation.

Each row represents the **animationSet** attribute that the sprite can hold, for example, if the character is walking forward or down, the **animationSet** will be updated to a string like '**walk-down**'. Most of these animation sets have about 4 frames to iterate through.

To achieve this animation illusion we need to define the **animationMap** attribute, something similar to the following:

The direction and movement will both be stored in a dictionary as an attribute of the sprite class called the **animationsMap** as I have mentioned previously.

Animation Algorithms

To completely achieve the animation feature, I will need to write algorithms for the updateSpriteSet() and the updateFramesLeft() methods.

```
METHOD updateSpriteSet(direction)
    // Reset the number of frames left to draw
    SET this.animationSet TO direction
    // Set the current animation set to the player's direction
    SET this.framesLeft TO this.framesLimit
END METHOD
```

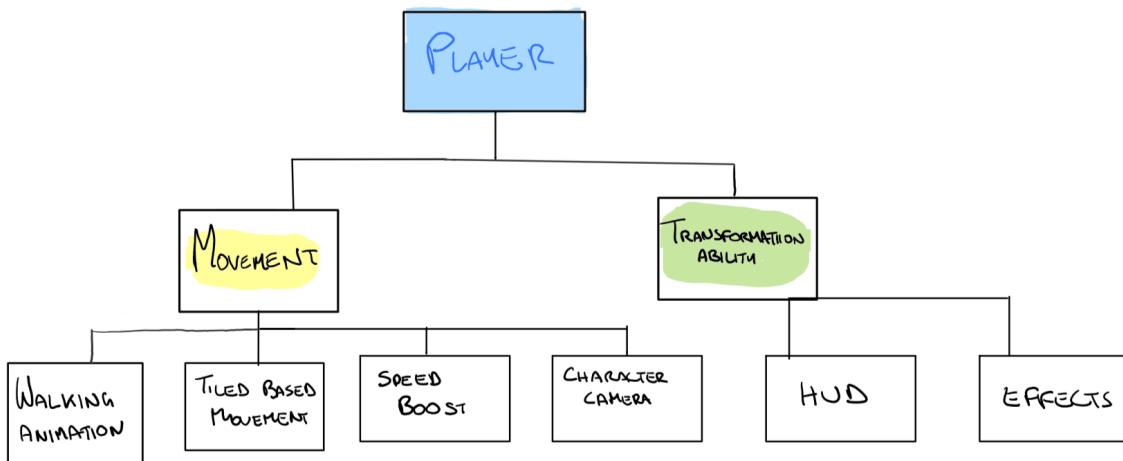
The **updateSpriteSet()** algorithm ensures that the entity's **animationSet** is updated to correspond to the new direction of the character, after this process, the **framesLeft** attribute is reset meaning that the new animation is going to be iterated instead.

```
METHOD updateFramesLeft()
    // If the animation has frames left to display
    IF this.framesLeft > 0 THEN
        // Decrement the framesLeft by one
        SET this.framesLeft TO this.framesLeft - 1
        RETURN
    END IF
    // Set the framesLeft to the framesLimit
    SET this.framesLeft TO this.framesLimit
    // Go to the next sprite frame
    SET this.currentSpriteFrame TO this.currentSpriteFrame + 1
END METHOD
```

The **updateFramesLeft()** algorithm is responsible for managing the animation and updating the frames to create an illusion of motion. This method begins by checking if there are any frames left to draw, if there is, it will decrement the **framesLeft** attribute meaning that the animation is progressing. If there are no frames to update then reset the counter by setting **framesLeft** back to the **framesLimit**, then which advances the sprite animation to the next frame.

This algorithm is simple and effective. I could use an alternative approach such as using libraries instead of using these complex algorithms, to simplify the game and allow me to use advanced techniques for animations. However, the original algorithm is far better than the alternative because it is more flexible such as allowing me to configure the speed of the frames which will be really essential for developing other features such as the speed boost feature. Moreover, this algorithm will aid me to develop the complete movement system, as without any animation, the movement will not look appealing to the users.

Player Class



The player class that I have decomposed using a hierarchy chart contains mainly two components; Movement and Transformation Ability. The player class will still have the same components, attributes and methods from the entity class, however, it is going to have some extra logic and overwrite some methods.

One of the methods that the player class is going to overwrite is the movement logic and methods from the entity class. The character movement methods are going to differ from the entity class. Where the entity instance relies on behaviour loops to dictate their movement patterns, the player instance will rely on the user's input keys to ensure that the character moves appropriately and responds to the user's input. The player class will also support a speed boost feature which requires a user input. This speed boost feature won't be required in the entity class because the user is not going to control an NPC or monster in this game, whereas in the player class, the user can toggle speed boost on and off with their input key.

The second feature that I am going to add to the player class will be a special ability feature. I have decided to make the special ability feature a transformation ability, where the character is able to transform and switch between monster and a character. This feature is only available in the player class, alongside the transformation ability, I will also need to add HUDs (Heads-up display) and effects to solve this problem effectively, as seen in the hierarchy chart above.

Overall, the player instance is a child class from the entity class, more suited for the user to control and perform actions as it offers additional features such as controlled movement and transformation ability.

Player Class inherit from Entity Class

Attributes:

- isPlayer = true
- Abilities
- HUDs
- Effects

Methods

- Update()
 - SpeedBoost()
 - SpecialAbility()

Attributes

The player class will inherit all the attributes from the entity class, but is also going to include a few more additional attributes such as **isPlayer**, which will allow the game to identify which one of the various entities is player class. Moreover, the player class will also include the Abilities attribute which will provide the player's method with information and instructions to execute the desired actions and special ability of the player.

As mentioned previously, the special ability

that I am going to develop will be the transformation ability, where the player is able to switch between two different entities.

Therefore, the **Abilities** attributes will be a dictionary which will store the information for their transformation ability such as the sprite of the transformation entity etc.

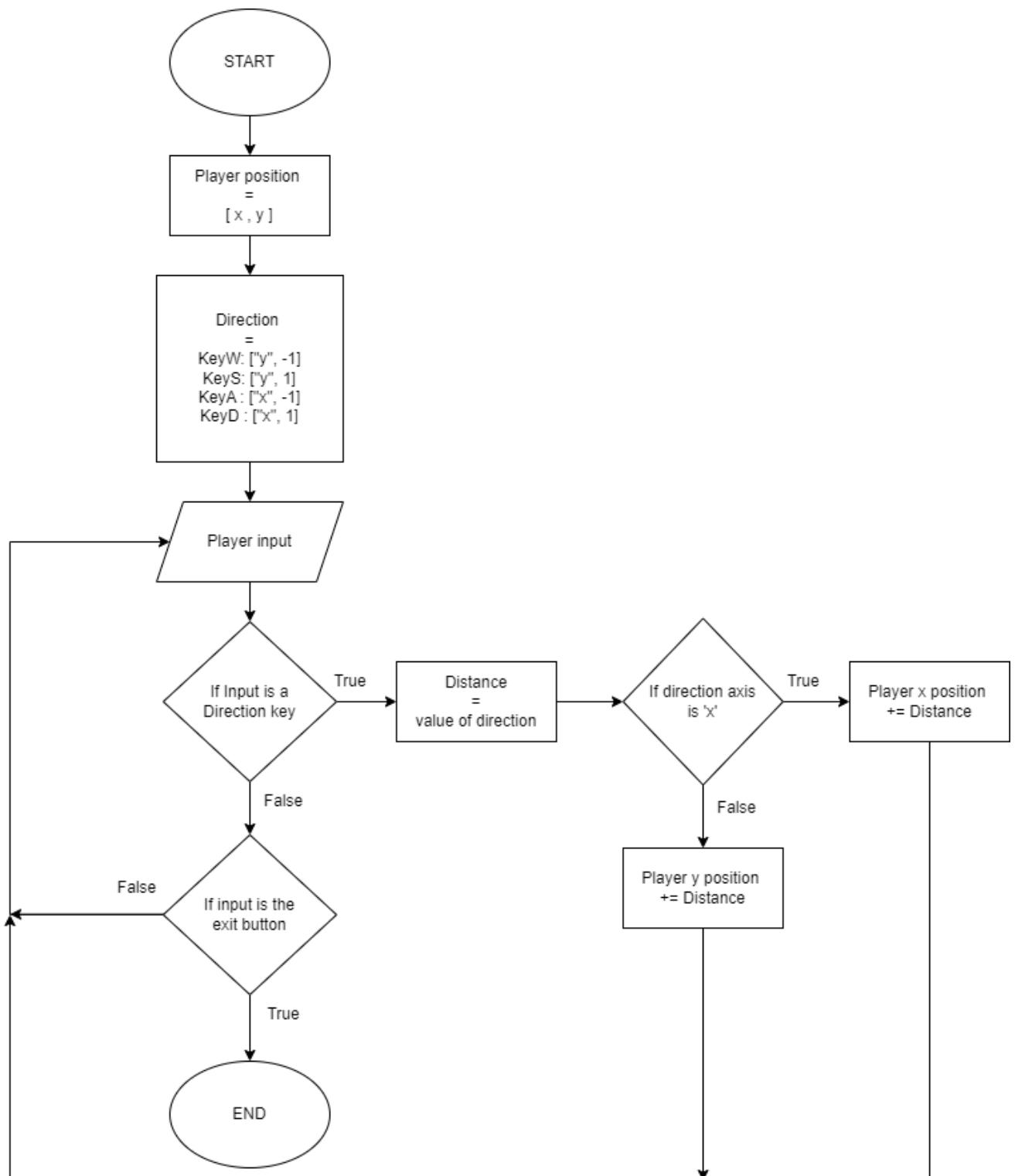
Because of this additional functionality to the player class, the game will also require **HUDs** and **Effects** to complement the transformation feature. As a result, I have added the HUDs and Effects attributes which will be a dictionary of instances. For example, the Effects attribute will contain a dictionary with a few effects instances, which has been constructed using the effects class, this will allow the player to access different effects.

I could have not stored the HUDs and effects attributes as dictionary objects, but rather a single HUD for the HUDs attribute and a single effect for the effects attribute, meaning that each attribute will store one instance of HUD and effects, which will simplify the game and rely less on global variable dependencies, as most other instances are global variables for easier accessibility. However, this would make the use of OOP useless because classes are built for the purpose of having multiple instances and not only one, furthermore, allowing the player to access multiple different HUDs and Effects will increase the game's functionality leading the user feeling more engaged with the game.

Methods

The player class is going to inherit all the methods from the entity class, except for the update() method, which the player class is going to override it as the player class requires to handle additional features such as speed boost toggle, transformation, controlled movement etc, These functions could all be implemented into one function.

Controlled Movement Flowchart



I have created the following pseudocode flowchart to make the development of the controlled movement feature more feasible. This feature, as seen in the flowchart, will allow the user to move and control their character in 4 different directions, each direction with their corresponding key.

This pseudocode flowchart initiates by defining two variables, the **player position**, which stores two key important variables; the **x and y** coordinate value position of the player entity.

Secondly, I have defined the **Direction** dictionary which consists of direction keys; W, A, S, D, which are stored as the key elements in this dictionary, where each key stores a list. Each one of these lists stores an axis and a value to add and subtract from the player's coordinate later on.

The game then detects any key input from the player, and checks if the input is found in the direction dictionary that I have declared previously. If the input is in the dictionary, the algorithm will then fetch the distance the player needs to travel which is either 1 or -1 depending on the direction, these values are then added to one of the player's coordinate axes again depending on the direction. For example, if the player presses keyW, the direction is "up", therefore only the y axis should be updated in this case, so the algorithm takes the direction value for the "up" direction, which is -1, and this value is then added to the player y coordinate, signifying that the player new position. This is valid because in the scope of this project, if you travel up, your y axis should decrement, and if you travel down, your y coordinate should increment, same is true for the x axis, where if you travel right, your x coordinate increments, and if you travel left, your x coordinate should indeed decrement. I have decided to approach the axes and directions of this game in this approach because the top right corner of the map should always be (0,0). However, this approach may not be suitable for maintenance in the future as it can be considered not mathematically appropriate. I could have used a more 'mathematically correct' approach as we see in most graphs nowadays, which is if you go up along the y axis, the y value should increase, however, I am more inclined to use my approach because I want to avoid any negative numbers and negative coordinates which may cause more errors in the future development phase of my project.

Once the code has updated the player's position for one game loop iteration, the algorithm goes back to detecting any inputs, however, there is a second scenario other than the direction input which can take place and will lead to the end of the flowchart. If the input is not in that dictionary, then the flowchart will check if the input from the player was an 'exit' input, meaning that the player is trying to exit the game, if that is the case, then the flowchart will end. While the game is running, these combinations of processes in the flowchart will always be executed as the methods for this algorithm will be always called in every iteration of the game loop, allowing the player to always travel whenever they want, making the game feel more responsive, interactive, engaging and increases that sense of freedom within the gameplay.

Controlled Movement Algorithm

```
// Define Player class that inherits from Obj
CLASS Player EXTENDS Entity
    // Define constructor function with configuration argument
    PROCEDURE constructor(config)
        // Inherit from parent class
        CALL super(config)

        // Assign movement direction for each direction
        SET this.directionDict TO {
            "up": ["y", -1], "down": ["y", 1],
            "right": ["x", 1], "left": ["x", -1]
        }
        // ...
    END PROCEDURE
```

This code snippet creates a new class which inherits from the Entity class as we have previously mentioned. This class takes in all the attributes from the Entity class and takes in a set of parameters using the config object, which will be used to initialise the player instance. For this controlled movement algorithm, we need to define one important dictionary which is the **this.directionDict** attribute, which maps each of the four possible directions to the corresponding axis and movement value for that direction. For example, if the player walks to the up direction, the player's y coordinate will be decremented by 1, as the value is negative. However, to execute this logic we will need some methods within the same class, I have proceeded to do so in the following algorithms:

```
// Define update method to commit changes to player
METHOD update(state)
    CALL updatePos() // Update position based on current direction

    // If a new direction is given, update direction and tile count
    IF state.arrow EXISTS THEN
        SET this.direction TO state.arrow
        INCREMENT this.TilesLeft BY 1
    END IF
END METHOD
```

One of the methods which is required for the controlled movement algorithm is the **update()** method, which will call **updatePos()** to execute the tile-based movement logic. The update() method takes in a **state** object which is responsible for committing changes to the state of the player, this method checks if the new direction set in the state object exists, if so, the game will update the player's direction to the new direction from the state object then proceeds to increase the **this.Tilesleft** attribute by 1 suggesting that the player needs to travel one more tile.

```

// Define updatePos method to update position based on direction
METHOD updatePos()
    // If the player still has tiles left to move
    IF this.TilesLeft > 0 THEN
        // Get axis and value for current direction
        SET [axis, value] TO this.directionDict[this.direction]

        // Update player position on the correct axis
        SET this[axis] TO this[axis] + value

        // Decrement tile count to indicate movement
        DECREMENT this.TilesLeft BY 1
    END IF
END METHOD

END CLASS

```

The **updatePos()** method ensures that the player's position is updated with respect to the user's movement key input. This method is only used when the player has tiles left to travel, meaning that the **this.TilesLeft** attribute is greater than 0. This method creates new temporary variables such as the axis and the value, which stores the information from the current direction using the **directionDict** attribute we defined previously. This updates the player's position on the corresponding axis. Then, the **this.TilesLeft** is decremented by one, meaning that the player has now travelled one tile after their position has been updated. Everytime the user inputs a direction key, these sets of methods are executed in order to ensure that the user is able to move their character to their desired tile and direction.

I could not override the **updatePos()** method and use the same method from the Entity class, as the Player class already inherits the methods and attributes from the Entity class, this would reduce the length of the code and unnecessary complexity and increase reusability and efficiency. However, the entity class's **updatePos()** method will be more suited to NPCs and monsters as it will require additional code which can handle behaviour loops, whereas the **updatePos()** algorithm for the Player class is suited to handle controlled movement.

Controlled Movement Usability



OR



As you can see in the following diagrams, if the user either holds one of the WASD keys or the arrow keys, the player will move corresponding to the input with the correct frame-by-frame animation.



OR



This is a common approach in most RPG games to resolve the user control problem, where the user is able to control their character as they desire by inputting the correct key.



OR



The user is able to travel in different directions as they are offered 2 keys for each direction, the WASD keys or the arrow keys.



OR



I could just add one set of keys to keep my game simplistic and consistent, however, different players will have different preferences and restricting the set of keys they can use may damage user experience due to the lack of preferences in the game, therefore it is better to add the 2 most common sets of movement keys, the WASD and arrow keys, as I have mentioned before and seen in these diagrams.

Game Battle

Most RPG action games include a **battle system** whether it is; first person shooting, tri-attack battle system, versus fighting system etc.

Turn-based battle system is also a good option and some successful games combine RPG games with this combat system.



based combat system

I will try implementing my own sort of game battle in my game, where the user can battle other enemies following the similar turn-based combat system. This allows my game to feel more strategic and gives an additional layer of complexity allowing the user to use more features, adding more variety to the game.

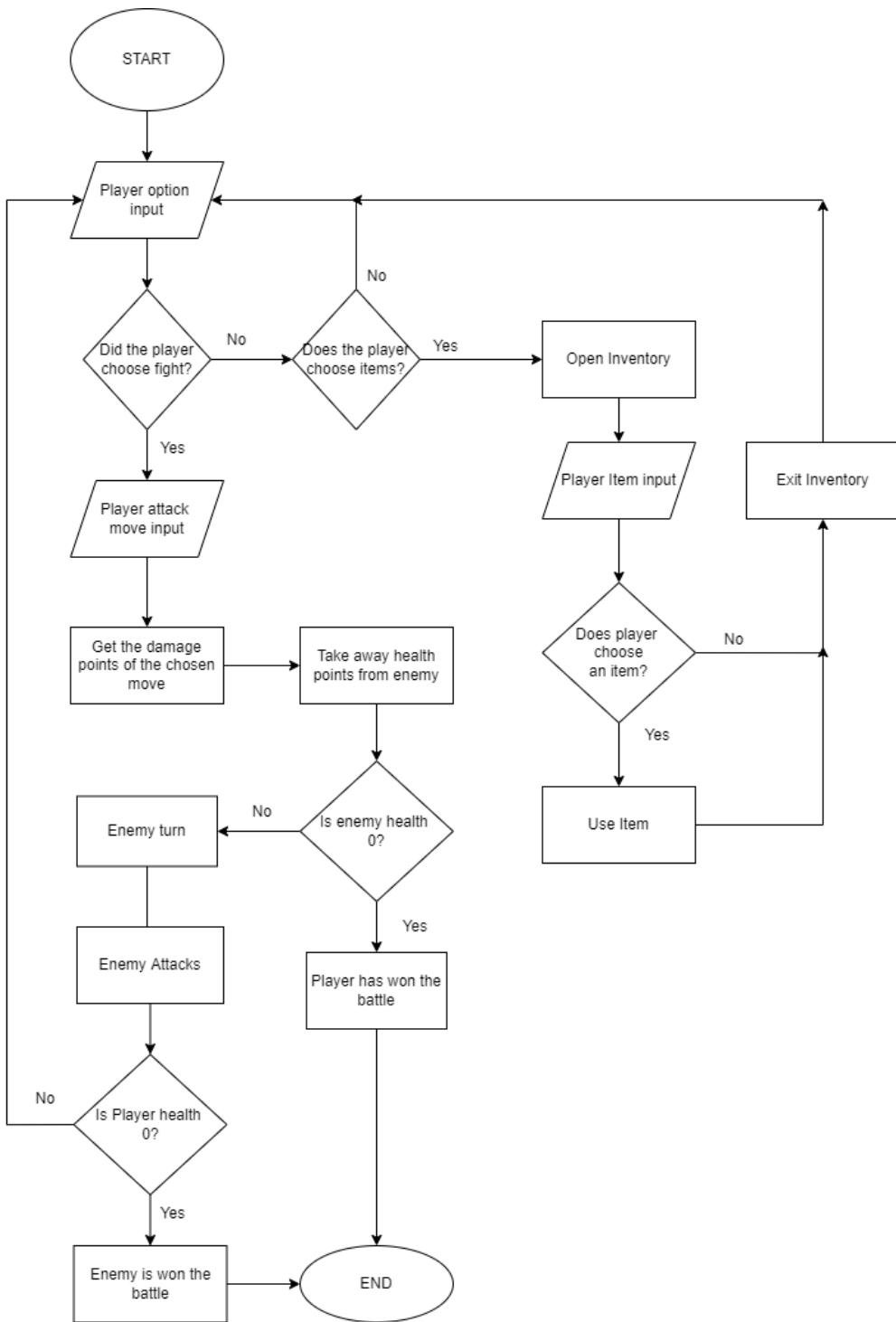
I could use an alternative common approach for most RPGs such as the Real-time battle system, where a battle takes place in real-time combat with the enemy, allowing the player to control the character's movement and perform more fighting actions. This can provide a dynamic gameplay experience, whereas the turn-based battle system can often take longer to complete as the player must wait their turn to attack which may not appeal to some players who prefer fast-paced action. However, a real-time battle system may be less accessible to players who struggle with quick reflexes. The turn-based system does rely on the player's reflexes and gaming abilities, but rather their strategy as the player must carefully consider their moves and abilities in order to defeat their opponent, this approach is much easier as I can configure the difficulty of the enemy easily by changing the damage that the opponent can yield, how much health does the opponent have etc.

Turn-based combat system is where two or more entities take turns in combat actions such as healing, attacking, defending and using items, the first entity to reach a health level of 0 loses. This is known as a strategic battle.

This type of combat system requires an intelligent AI as an opponent that needs to decide which move would be suitable in which scenario.

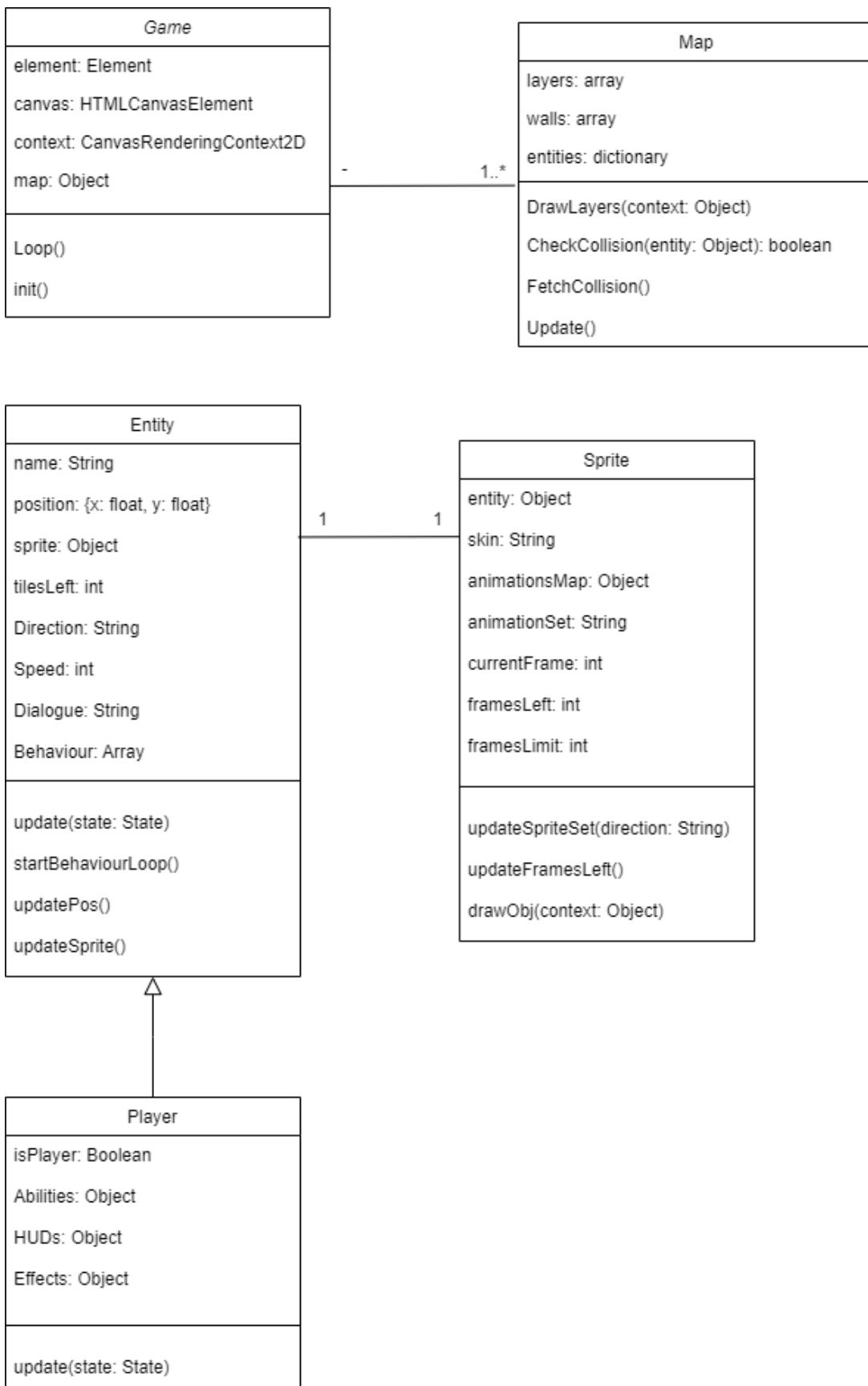
Here's an example taken from *Pokemon* of a turn

Game Battle Flowchart



Here's a possible flowchart design that I may implement in my game, however, this feature may not be totally feasible due to the short period of time that I have got on my hand, and the amount of extra additional features it requires make a complete battle system, for example, an inventory system, a health system, a power system, a completely new GUI etc. However, if I do have enough time and resources needed to implement this feature, it is always good to have a flowchart system to look back on and use it to guide the development.

Unified Modelling Language



This system describes the system made up for various classes and shows the relationship between these classes and how they relate to create the entire structure of the game. The most important class that we already explored before is the Game class which has attributes needed to draw the elements onto the canvas. This class always contains the map attribute which will store the current map instance, as a result, the game class will share a association link, where the game can store more than one instance, because the game class will be able to switch to different map instances, which will facilitate for the development of certain features such as map transitions, which is common in all RPG games similar to mine. Since my project will only have one game, the map class won't be required to create any instances for itself, therefore the multiplicity between the game and the map is one-to-many, where each instance of map is associated with only one instance of the game.

Furthermore, the map class will handle the game's layers, collision walls and entities for its attributes, and will include different methods which also involve this particular attribute such as updating the game world, drawing layers and handling collisions between the entities and the map walls.

The second part of the UML system focuses around the players, entities and the monsters, which you can also consider to be interactive map objects. The entity class is the parent class of the player class, both classes will have similar attributes and methods, with the only exception being that the player class will have additional features such as controlled movement from the user, which will be implemented in the **update()** method, abilities, HUDs and effects.

The sprite class is a bit different from the player class and the entity class, whereas this class is responsible for the graphics and spritesheets of different entities, it is also required to handle the animations of the entities, such as their walking animation or transformation animation for the player class. This class contains one of the most vital methods for entities in this game which is the **drawObj()** with the sole purpose of drawing entities.

The sprite class shares an association link with the entity, with the multiplicity between a one-to-one relationship with the entity class.

I could have used an Entity-component-system architecture which separates entities into their essential components and then systems are created to process each component. This approach may be better than the UML system because it can provide enhanced performance and increased scalability, however, the UML approach is more intuitive, especially for OOP based games such as mine, this approach of using a UML will allow me to easily understand the hierarchy of relationships and develop the components of this system. Whereas the ECS architecture is not as compatible with OOP and will require me to think in terms of components and systems rather than classes, which will make the development process harder and unnecessarily lengthier.

Testing

Testing will be an essential aspect in the development of this project, where a complete test plan is required to ensure that the game is of high quality, robust and satisfies the expectations for the users and stakeholders. I am going to use an iterative testing strategy to approach which involves a repeating set of testing over multiple iterations.

This approach allows me to make small changes for every test iteration ensuring continuous enhancements during the game development process. This will increase the robustness of my game because it is being thoroughly tested for every change I make to the code, and maximises user satisfaction. This approach also allows the developer to collect input from stakeholders which can be used to improve the game, ensuring that the game meets expectations to satisfy the target audience by incorporating stakeholder feedback into the testing process.

The main purpose of this iterative testing strategy is to test for the following errors:

1. **Syntax Errors:** Can occur when the syntax or the structure of the code does not follow the rules of the programming language, they do not need any type of plan, therefore, I will not be mentioning these types of errors as they are easily solvable. They are also easy to see as the IDE will fail to compile and will reference to the syntax error present in the code
2. **Logical Errors:** Errors that may occur when the code does not output the desired output or behaviour. Because the code can still operate without error messages from the IDE, they are harder to detect and requires to be recorded in the development section
3. **Runtime Errors:** Errors that may occur during code execution as a result of an unanticipated input. They will usually cause the programme to crash and may also deliver unprecedented outcomes, one common runtime error that I may encounter in this project may be array index out of bounds.

Validation

There are four types of validation rules that I will need to consider during the process of the iterative testing, the 4 are:

1. **Normal validation:** test data that falls within expected or standard input
2. **Invalid validation:** test data that is outside of the validation rules of input values.
3. **Boundary validation:** test data that checks the maximum/minimum input values
4. **Erroneous validation:** test data that intentionally contains errors to check for its robustness.

It is vital to test for these types of errors and different validation rules, so I can ensure that the code is functioning and satisfies the project requirements and success criteria. It will allow me to identify and record any flaws before the code is released, ensuring that the code is of excellent quality, bug-free and the user has a favourable experience with this game.

Test No.	What I am testing	Tested Code	Result Required
1	Check if the WASD keys allows the player to move	playerMovement()	The player entity must move and travel in different direction corresponding to the input keys

Example

Here's an example of a test plan, this hypothetical test checks if the player is able to move using the WASD keys, which is a type of normal validation rule. I will be using this specific format for testing all my main features and algorithms throughout the development process.

Overall, using this strategy for testing will allow my game to be of good quality and meet stakeholder expectations. Furthermore, it will allow me to identify any issues and bugs before releasing the game. I will be using this strategy on individual components of the code, such as specific algorithms, classes and methods rather than the whole game at once, making sure that every single component works perfectly on its own first, this will facilitate the post-development testing as all the components will be bug-free before testing the whole system in later stages.

I could test the software using a different strategy, such as waterfall testing, which is a linear approach to testing in which each phase of testing must be completed before going on to the next. The software requirements and the development process should both be clearly specified for this technique to work. However, iterative testing is significantly better for creating a 2D RPG game since it includes testing the software in brief sprints, which enables me to quickly identify and record any issues before moving on to the following stage of the development process. Iterative testing will also allow me to receive continuous feedback, make adjustments and allow for increased flexibility throughout the development process. Therefore, I believe that iterative testing is the best way to test my changes when developing a complex and dynamic game like my proposed solution.

Here's the main tests for the iterative testing for the development process, I have created a similar test plan as seen in the following table:

Feature to test	Test Data / Functions / Input	Justification
Check if the game loop iterates consistently and free of errors	GameLoop()	Ensures whether the game will be capable to carry out iterations consistently to provide a smooth gameplay and fewer bugs.
Check if the canvas can draw maps correctly	map.collisionLayer map.upperLayer map.lowerLayer map.drawLayers()	This test is important because it ensures that the map are drawn in the correct order which will allow to identify collisions more easily later on
Check if the player can travel specific number of tiles and also display the animation correctly	Player.tilesLeft Player.update()	Ensures that the player grid based movement is working correctly facilitating the development of user controlled movement for the player class
Check if the user is capable of navigating the player entity	keyInput.directions.init() Player.update() W,A,S,D keys Arrow keys	Ensures that the user is able to navigate the player instance in order to play the game and have access to performing more actions.
Check if the player's movement speed changes if the user toggles the speed boost ability	Player.speed keyInput.speedBoolean Player.update() ShiftLeft Key	Ensures that the player can travel faster with the correct animation and speed corresponding to the test data using the Player.speed attribute.
Check if there is a character transformation ability with the correct HUD and effects	Player.transform Player.update() HUD.drawHUD() FX.drawEffects()	This test is important because it ensures that the player is able use the transform ability without any errors or restrictions and ensures that the effects and HUD are drawn and updated correctly
Check if the collision works properly between different entities and map walls	map.checkCollision()	A crucial component for most games that ensures that the player has limits and boundaries of where it can travel to.
Check if the player can navigate between different maps	Game.map.exits GameLoop()	Ensures that the player is able to explore different maps which is the main objective of this game.
Check if NPCs have their own behaviourLoops and interactive features such as dialogues.	keyInput.selectBoolean map.playerInteraction() Obj.update() Enter key	Ensures NPCs serve a common purpose by executing their behaviours properly and allowing the player to interact with them and display a dialogue as expected.

Post Development

In order to determine whether the game functions properly after development and iterative testing is successful, I can now move on to the post development testing. There are a variety of ways to conduct this stage of testing:

Stakeholder testing:

Where I ask each stakeholder who has taken part in aiding the development of this project by providing feedback and input to test out the final product. This allows them to access the game and test it out, as this test aims to receive feedback from stakeholders about the game, if there are any glitches, or if the requirements have been met correctly or what additional changes should be made. This helps to ensure that the final game meets the expectations of the stakeholders, which is essential for the success of this project.

Functional Testing

It aims to verify the features of the game to ensure they are working as expected and according to the success criteria and requirements by testing these functionalities.

Test No.	Feature to Test	Input	Expected Result	Justification
1	Player movement	WASD/arrow keys	Player moves in the correct direction, animation and correct camera.	Ensures that the player is able to control an entity with all the movement features combined.
2	Collisions	WASD/arrow keys	Player cannot move through walls	Ensures that the player is restricted from travelling through objects
3	Dialogue system	Enter key	Player can initiate dialogue with NPC	Ensure that the game's dialogue system is working correctly
4	Map navigation	WASD/arrow keys	Player can travel to several different maps	To ensure that the player can explore and travel to different maps in the game
5	Transformation & Speed boost ability	ShiftLeft Key	Player switches to a monster entity and moves at faster speed.	Ensures the player is able to transform into a monster entity and increase their movement speed

Robustness Testing

Where the game is tested for their ability and capability to handle unexpected or invalid inputs, which aims to ensure that the software does not produce any unexpected errors, crash, and ensure that the game remains stable under adverse conditions.

Here's a test plan table of all the robustness tests that I am going to carry out:

Test No.	Test Description	Test Procedures	Expected Result	Justification
1	Test if the game is able to handle interruptions during when the user holds a key and executes a task from outside the game simultaneously	Attempt to hold shift key and then right click to toggle on the usual context menu	The game should be able to detect the interrupt and avoid the “stuck key” issue	This tests a common issue in most web-based games where the game gets stuck on the last input key when the user interacts with something outside of the canvas during the gameplay
2	Test the robustness of the game when running for long periods of time	Leave the game running for at least 1 hour	The game should work as normal even after running for an extended amount of time	Ensures that the game can handle prolonged usage without impacting the stability of the game's performance
3	Test the robustness of the game when tested under a unstable internet connection	Simulate network issues using a network limiter at varying different speeds, then test out the game and its functionalities.	The game should be able to handle really slow internet connections due to its small scale	Ensures that the game can be playable for players with connectivity issues
4	Test the robustness of the game when the website is refreshed at really fast rates and repeatedly	Attempt to spam click the refresh button	The game should load, not crash at all times and initialise itself for every refresh	Ensures that the game is able to handle high numbers of refresh requests and still proceed to display the game correctly.
5	Test the robustness when the user spams a key	Spam all the valid keys on the keyboard during the gameplay	The game should work as normal without crashing	Ensures the game's robustness and ability to handle rapid, repeated and high number of input and requests

Structure of Solution Development

After defining all these features, classes, flowcharts, algorithms and tests, I can now start the overview for structuring the development of my game. I have only included the main features to develop, without taking into account sub-features in order to simplify the structure

Structure	Description	Justification
Canvas setup	Creating and setting up the game canvas and styling the canvas accordingly to the design and the GUI	Necessary for creating a visual representation of the game and user interaction with game elements
Game Class	Managing the game mechanics and game loop which will be the core method of the whole game	Provides a centralised location for managing game logic and ensures that the game runs smoothly and correctly without errors.
Map class	Will be responsible for drawing all the maps with the correct entities and collisions	Provides a structured way to define the game world, its elements, and boundaries, which the player can interact with
Entity Class	The main class for all entity objects such as the player, NPCs and the monster	The game world's characters, monsters, and things are defined by entities, making this really essential for developing other features
Player Movement	Allowing the player to move around the game world with a correct sprite animation, speed and camera	Allow the user to control a player instance which allows them to perform more actions such as exploring the map and interacting with other entities, which is essential.
Character Transformation	Allowing the player to transform into different characters	Gives the game more gameplay variety and depth because allowing the user to perform more actions, which is important
Collisions	Handling collisions between objects, maps and boundaries	Ensures that entities interact with one another realistically and stops them from overlapping or going through one another.
Map Navigation	Enabling the player to switch between various maps	Enables the user to explore between several game world maps, diversifying the gameplay experience, which is the main purpose of the game.
NPC Behaviour & Interaction	Defining the behaviour movement patterns entities and how the player interacts with the NPCs	Enhances the game's immersion, gives entities a purpose, and makes the game feel alive which is essential.

DEVELOPMENT

Canvas Setup

```

<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>2D RPG</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
</head>

<body>
    <!-- The game div is going to contain the display of the game -->
    <div class="game">
        <!-- The game canvas -->
        <canvas class="canvas"></canvas>
    </div>

    <!-- Scripts -->
    <!-- Links all the scripts back to the HTML file-->
    <script src="init.js"></script>

</body>
</html>

```

The following HTML file is used to launch my game, it creates new divs inside the body where it indicates where the game will be shown. Furthermore, I will need to add CSS styling to the different divs, therefore, this approach seems more flexible as setting HTML classes and subclasses for divs will allow me to style each div individually.

Moreover, I will be adding a lot of JS files throughout this project, as a result I need space to link all the scripts so the js files could be initialised. I've made a brand-new file called "init.js" that will start up my game.

Canvas Style

```

* {
  box-sizing: border-box;
}

body {
  /*Declares the background image for the website*/
  background-image:
  url('https://allfreedesigns.com/wp-content/uploads/2015/06/black-patterns-11.jpg');
  padding: 0; margin: 0;
  overflow: hidden; /*Ignores scroll responses to arrow keys*/
}

.game {
  position: relative; /*Canvas is relative to the parent element*/
  width: 1280px; height: 700px; /*Aspect ratio for the game canvas*/
  margin: 0 auto; /*centers the canvas, the position is set
  automatically based on the browser*/
  margin-top: 2%; /*Descends the canvas downward by a few pixels*/
  outline: 3px solid red; /*Outline of the game canvas*/
  filter: saturate(1.5) brightness(0.75);
}

.game canvas {
  /* the images inside the canvas is going to be rendered and
     scaled by multiplying pixels*/
  image-rendering: pixelated;
  background-color: #141B1B;
  width: 1280px; height: 700px;
}

```

The css code above configures the style of my game and game canvas classes.

At the start of the CSS file, I have set box-sizing to border-box which means the padding and borders of every element won't affect the box, which is an imaginary outline used to calculate the position and size of everything in a website. I could use other alternatives such as 'content-box' instead of 'border-box', where the browser will calculate the size of a box by including the padding and border of the element.

However, 'border-box' is the best option because it allows for more precise calculations leading to better user experience.

The body configures the style of every element and div within it, since I am planning to only add elements inside the body, the following set of styles affects every element and div in my game. I started by setting the background image, which is the background outside of my game canvas. I followed it by setting padding and margin to 0, meaning there will be no spacing between different elements within my `<body>` element. This allows me to position every element exactly where I want it, whereas setting padding and margins more than 0 will not allow me such flexibility.

Moreover, I set overflow to hidden which prevents the page from scrolling up and down when the arrow keys are pressed, these sets of keys are really important later on as it will be used for player movement. If I don't set overflow to hidden, everytime the player will move their character, the website will scroll up and down which will affect user experience negatively.

I have added style configuration for my game class. I set the position of the game class relative, meaning that it will be affected by the position of the parent element, which is the `<body>` element. Furthermore, I have added the size and resolution of the game, and the position where I set margin as 0 auto, which will automatically centre the game relative to the browser's position. I also added extra configuration to the game element class by descending by 5% to better ameliorate the position. Finally, I added an outline temporarily for testing purposes as it will allow me to debug and test the elements even easier as it highlights the borders of the game class.

Finally, the game canvas is the class element which will display the game and all the draw functions will be referenced to this canvas. As my game is a 2D pixelated theme game, I have set the image-rendering to "pixelated", meaning that if I scale any image or graphic element and draw it inside this game canvas, the following image will be scale using the nearest neighbour implementation which is the simplest and fastest image scaling technique and best suited for pixel based games like mine. I could use other complex variations of scaling such as bilinear which is the most common, however, it will make my game blurry which will significantly affect user experience negatively and the image rendering won't be clear.

Test

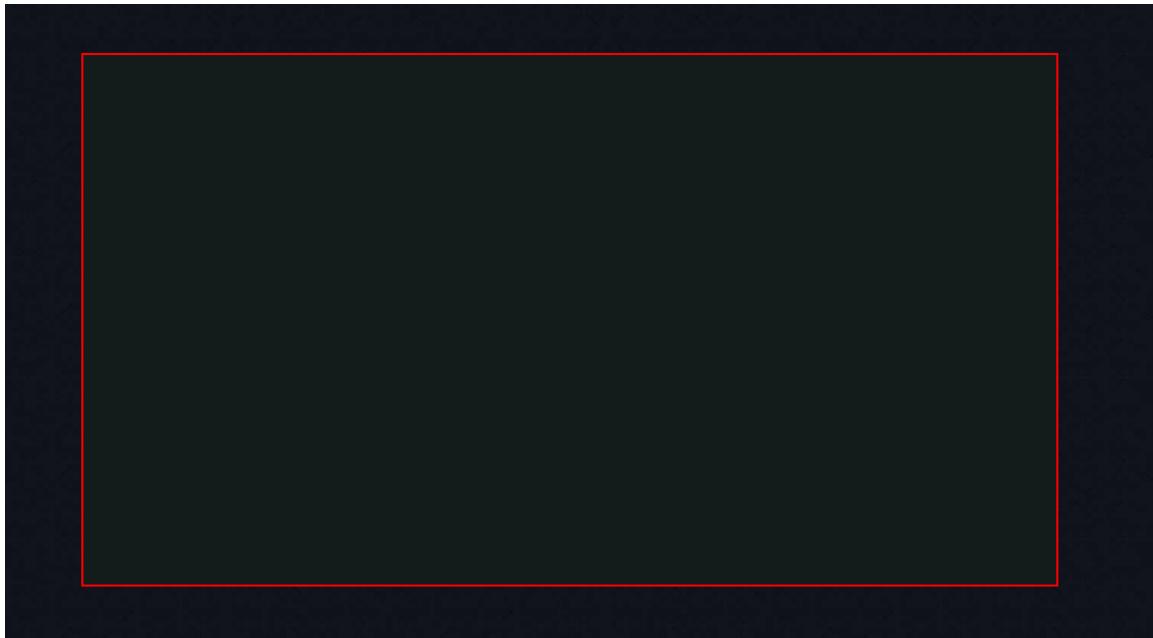
Here is my first test plan where I am going to check if the canvas is displayed and draws the images perfectly without any errors.

Test No.	What I am testing	Test code	Result Required
1	If the game canvas displays correctly	N/A	The canvas should draw with the correct CSS
2	If the game canvas displays pixelated images correctly	N/A	The canvas should be able to display the same image in the correct position with different sizes without losing resolution

The first test is a **normal** validation test ensures that the game canvas is accurately presented, I have not specified the test code for it because it would be dependent on how the game canvas is implemented and requires a visual evidence such as a screenshot, the outcome of the canvas must meet all the requirements set by the CSS that I have coded in the **style.css** appropriately. This test is important because if the game canvas is not drawn correctly, then it will lead to limitations in developing other features into the game.

The second test is a **normal** validation test that ensures that the game canvas can display pixelated graphics without decreasing the resolution and quality, again the tested code is not provided because it requires visual evidence such as a screenshot. I am expecting the game canvas to be able to display the pixelated graphics in the correct position and size without affecting the quality of the image. This test is important because testing the quality of images will help me to ensure that the game is capable of handling pixelated graphics, which is essential as my game will be entirely based on pixelated graphics, if this criteria is not met, it will make my game blurry and less appealing to the user which will definitely affect the user experience negatively.

1st Test: First Iteration



The canvas is displaying correctly in the right centre position and even after resizing the browser. **The test is successful**

2nd Test: First Iteration

In this test, I am required to display an image onto the canvas. In order to do so I need to create a new file, called '**init.js**'. This file will initiate my whole game.

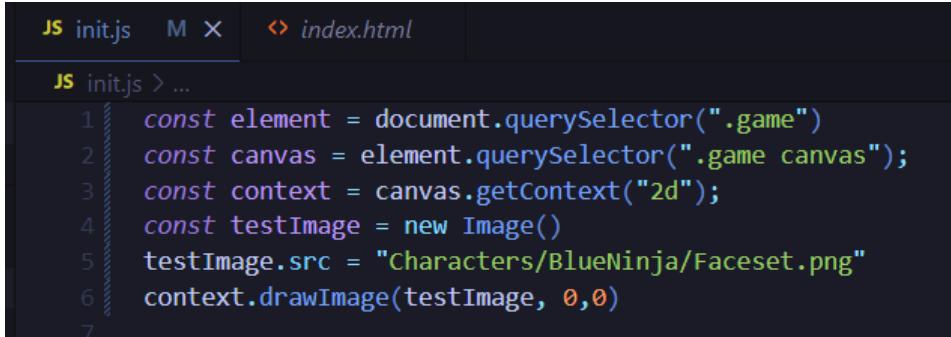
To start drawing on the canvas, I need to define the following:

Element: the document with the class "game" is chosen by element using the document.querySelector method. The element variable is then assigned the "game" class.

Canvas: This variable selects the first canvas element, the child element of "game" class; "game canvas" class. The canvas variable is then assigned the "game canvas" variable.

Context: To obtain the 2D rendering context for the canvas, this variable calls the getContext method on the canvas variable. Images and shapes are drawn onto the canvas using this context.

The following code is temporary and only used for testing purposes, I will use the three variables again in the future as I will need it to draw images onto the canvas.



```

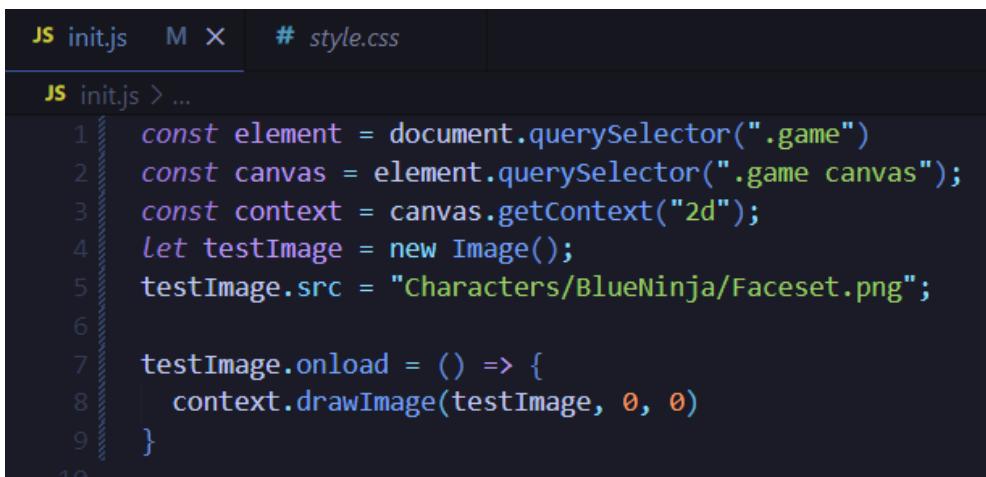
JS init.js M X index.html
JS init.js > ...
1 const element = document.querySelector(".game")
2 const canvas = element.querySelector(".game canvas");
3 const context = canvas.getContext("2d");
4 const testImage = new Image()
5 testImage.src = "Characters/BlueNinja/Faceset.png"
6 context.drawImage(testImage, 0,0)
7

```

The test was unsuccessful, as the results did not meet the requirements. The 'testImage' was not displayed at all and no errors were found, which led me to believe that the image was not loaded correctly when the `drawImage` was run.

2nd Test: Second Iteration

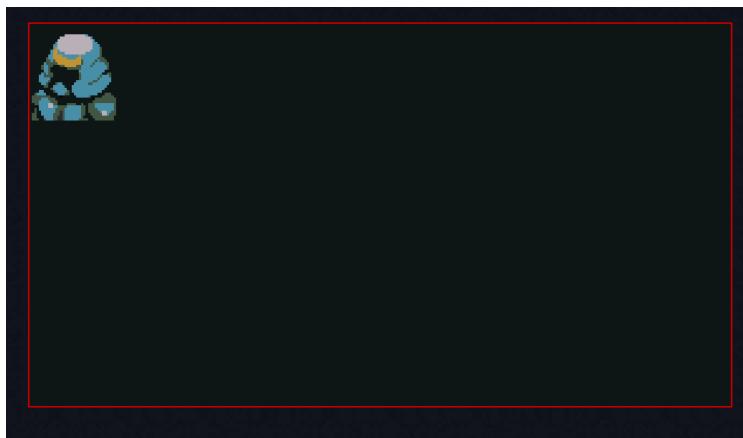
As a result, I implemented the `.onload()` function for this test. When the image is loaded, the code then will proceed to display the image at the correct position.



```

JS init.js M X # style.css
JS init.js > ...
1 const element = document.querySelector(".game")
2 const canvas = element.querySelector(".game canvas");
3 const context = canvas.getContext("2d");
4 let testImage = new Image();
5 testImage.src = "Characters/BlueNinja/Faceset.png";
6
7 testImage.onload = () => {
8   context.drawImage(testImage, 0, 0)
9 }
10

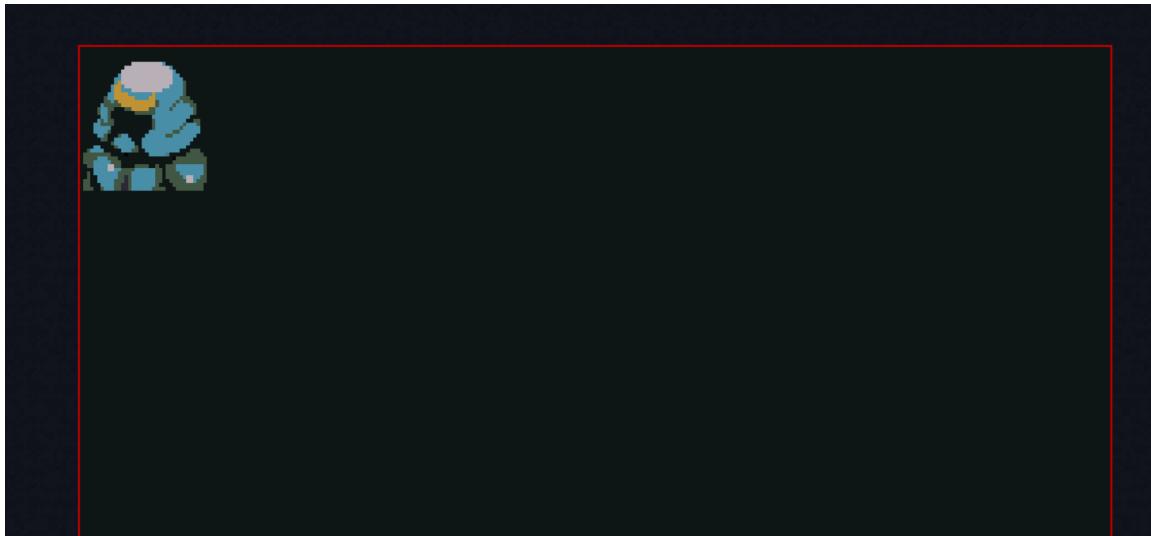
```



The test is partially successful, now I need to re-run the same test, however, this time I need to change the size of the canvas and check if the resolution of my image is deprecated or any other malfunction.

```
17 .game {  
18   position: relative; /*Canvas is relative to the parent element*/  
19   width: 1280px; height: 700px; /*Aspect ratio for the game canvas*/  
20   margin: 0 auto; /*centers the canvas, the position is set  
21   automatically based on the browser*/  
22   margin-top: 2%; /*Descends the canvas downward by a few pixels*/  
23   outline: 3px solid red; /*Outline of the game canvas*/  
24   filter: saturate(1.5) brightness(0.75);  
25   transform: scale(1.5) translateY(20%);  
26 }
```

In the CSS file, I added a transform attribute to the game class, which will increase the size of the game element by 1.5 and add spacing from the top so the top of canvas is visible post scaling.



The resolution seems to be constant after the canvas size has increased, meaning that the scaling process has undergone a nearest neighbour scaling. **The test is successful**

Game Class

For my game, I will need to include a class which acts as the ‘spine’ of the whole. This class will bridge every other class together, initiate and set up the game, and it will also control all the activity of the game, such as drawing images.

As previously mentioned, my game will be based on OOP, as the purpose of the game class is to act as a parent class and a central component that will contain every component of the game, including other classes.

I could use a different paradigm such as functional programming or procedural programming, where instead of creating a class, I could store all the functions in one file. However, I am planning to stay organised and cleaner as I will have lots of complex objects and methods. Furthermore, using OOP will facilitate testing individual components rather than testing every other component simultaneously, so is easier to maintain. Moreover, using OOP will also allow me to create multiple instances of one class, which will lead to cleaner and efficient use of code, as it reduces repetitive codes throughout my game.

```
// A game parent class which include every component of the game
class Game {
    constructor(config){
        this.element = config.element;
        this.canvas = this.element.querySelector(".game canvas");
        //Reference to the HTML canvas
        this.context = this.canvas.getContext("2d");
        //Will give us access to different drawing methods
    }

    //The init method will start the game
    init() {
    }
}
```

I have created a new file called “**Game.js**” containing a class called **Game**.

This **Game** class takes a **config** object as an argument which will contain the necessary data to create an instance of the game by filling in the attributes of this class. One attribute that this config object will store is the reference to an HTML element that will contain the game. In future development, I am going to add more attributes to the config object, however, for now I am only focusing on the initialisation of this game class.

I can now use the **this.element** attribute, which has been defined using the config object, to create the canvas and context, as these attributes will allow the game to draw graphics on the canvas appropriately. I have explained this approach and logic before when I was testing if the canvas was able to draw the images properly.

One of the necessary methods that I will need to implement onto the game class in order to initialise the game is obviously the **init()**. This approach is really common in most indie games, where one method starts off the game by initialising the game loop (which I will develop soon). The purpose of this method is also to create the necessary instances of classes needed to start the game such as initialising a map instance, entity instance, player instance etc. Overall, The class's goal is to offer a flexible and organised way to manage the game's various parts and elements. However, at the current moment it doesn't do anything and it requires an instance to be created and to run.

```
//Creates a new instance of the game
const game = new Game({
    //the game class is declared as the element where the game will run
    element: document.querySelector(".game")
});

game.init(); //this method will start the game instance
```

In the **init.js** file, I have created a new instance of the Game class called "**game**", I am planning to only use one game instance in the entire game, therefore there is no need to create other instances of the Game class. I have configured only one attribute of the game class which is the element which specifies in which HTML class the game instance will run. I have retrieved the element using the `document.querySelector` function.

To initialise the game, I will need to call the method of the game instance which is the **init()** method. Currently, the **init()** method is empty and won't initialise anything.

I could use a plain function instead of a class to create a game instance by using a plain function to define the game logic within itself and return a single function that will start off the game. This approach is simpler than using OOP, however, due to my project being complex, this approach will lead to inefficient use of resources and depreciate user experience due to the slow and poor performance, because it won't be capable to handle different interrelated components later on. Therefore, using class would make my project easier to maintain, organise and code.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the game instance has been created and the init() method works correctly	Game.init()	When the init() method is launched, the code inside of the method should execute. For example, if the code inside the init() method is <code>console.log("Working")</code> , then the console should print "Working" which will suggest that the game instance has been initiate correctly and the methods are working

This test, follows a **normal** validation rule which checks if the game instance has been created if the init() method works correctly, there is only one single test for this simple feature and the test input is simply running the game, the expected result is that when the **init()** function is called, the code within method is going to be executed if the method works perfectly. To carry out this test, I can use a `console.log()` method which when the game is run, whatever is in the `console.log()` function should print itself onto the console, which will indicate that the game instance works perfectly and is ready to be used for further development. This test is important because this init() method needs to work, if not, none of the other methods that I am going to implement are not going to run appropriately. As a result, the init() method is the foundation for additional testing and validates that the game meets the criteria to run other functions.

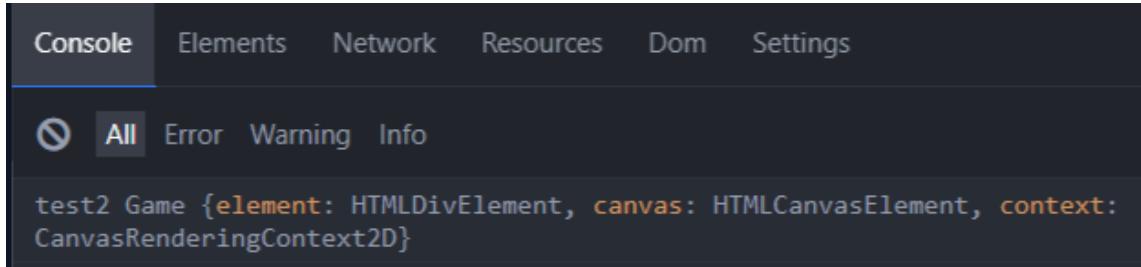
1st Test: First Iteration

```
// A game parent class which include every component of the game
class Game {
    constructor(config){
        this.element = config.element;
        this.canvas = this.element.querySelector(".game canvas");
        //Reference to the HTML canvas
        this.context = this.canvas.getContext("2d");
        //Will give us access to different drawing methods
    }

    //The init method will start the game
    init() {
        console.log("test2", this)
    }
}
```

The following test should display “test2” and the attributes of the game instance using the `console.log()` function that I have explained before.

Result



The game instance is working correctly with the correct attributes and the `init()` method works perfectly as well.

The test was successful

Drawing Layers

Now that our game class is working, we can start drawing images onto our canvas.

The game class will display the maps on the canvas, therefore I need to make a method where it takes an image and position and draws it on the HTML element which is the context attribute, in this case the “.game” element.

```
class game{
//...

//function that draws images on the canvas
drawMap(imageInstance, posX, posY) {
    //The parameters is the image object, x and y position that will be
    imageInstance.onload = () => {
        //Draw the image if the it is loaded
        this.context.drawImage(imageInstance, posX, posY)
    }
}

//The init method will start the game
init() {

    //Lower Layer
    const lowerLayer = new Image(); //creates a new image
    //The source is the attribute of the instance
    lowerLayer.src = "/Maps/starting house/lower layer.png";
    //Calls the method for drawing the image
    this.drawMap(lowerLayer,0,0);

    //Collision Layer
    const collisionLayer = new Image();
    collisionLayer.src = "/Maps/starting house/collision layer.png";
    this.drawMap(collisionLayer,0,0);

    //Upper Layer
    const upperLayer = new Image();
    upperLayer.src = "/Maps/starting house/upper layer.png";
    this.drawMap(upperLayer,0,0);

}
}
```

In the following code, I have added the drawMap method, which takes the image object, the x and y position of where it needs to be drawn as parameters. It then proceeds to draw the image provided onto the canvas once it has loaded.

I have also changed the **init()** method of the game class, where I have called the **drawMap()** method, as this function is called three times to draw the three different layers for the map. The “lower layer” is the lowest layer, where the player can walk on top of this layer. The “collision layer” is the layer where the player and this layer are on the same z-axis, therefore, the player collides with the image in this layer.

The “upper layer” is the highest layer which is on top of the player, when the player and the upper layer collide, the player will move underneath the layer which will give an illusion where the player just disappears but it is still drawn under the upper layer.

Test

Test No.	What I am testing	Test code	Expected Required
1	Check if the drawMap function works	Game.drawMap()	At least one image should be drawn
2	If the position of the image corresponds to the PosX and PosY	PosX, PosY Game.drawMap()	‘Test’ should be printed on the console
3	Check if the position of the map is outside the canvas, it should hide the excess section of the map	PosX, PosY Game.drawMap()	The excess section should not be visible
4	Check if the layer are ordered correctly	lowerLayer, collisionLayer, upperLayer, Game.drawMap()	The lowerLayer should be at the bottom, the upperLayer at the top and the collisionLayer in the middle.

The first test is a ‘**normal**’ test that determines whether the **drawMap()** function is operational. This test is justified because if the **drawMap()** methods fails to meet the expected result, the maps will be incorrectly drawn and presented to the user, which will make the game unplayable, at least one image should be drawn onto the canvas if this test is successful and meets the criteria set in the test plan table.

The second test follows a ‘**boundary**’ test whether the image’s position corresponds to their coordinates, which are obviously the PosX and PosY values. This test is important because if the position of these images are not correct, it will cause various errors and won’t allow me to add more images appropriately in future development.

The third test is also a '**boundary**' test where I will be determining if the map's position is outside the canvas , the excess area of the map should be hidden and only the area which is positioned within the bounds of canvas should be visible. I could not test and let the excess area of the map be visible, however, leaving it like this would disrupt gameplay, will cause further confusion and may even crash the game because I will need to add some large maps, which won't be possible to show every bit of these maps. If the whole map is visible, it will mean that the size of the map will be minuscule, as a result, it will remove important details from the game and the user will not be able to access many features or even find their player entity on the map.

The fourth test is a '**normal**' test which determines if the layers are properly ordered. If I don't test and ensure that the layers are ordered correctly, there is a high possibility the game map will not appear correctly. There could be a chance that the lower layer may be drawn after the upper layer, or the collision layer being drawn over the upper layer etc. This is obviously a self-explanatory flaw that must be prevented in order for the game to be playable and be consistent.

1st Test: First Iteration

```
class Game{
    //...
    //The init method will start the game
    init() {

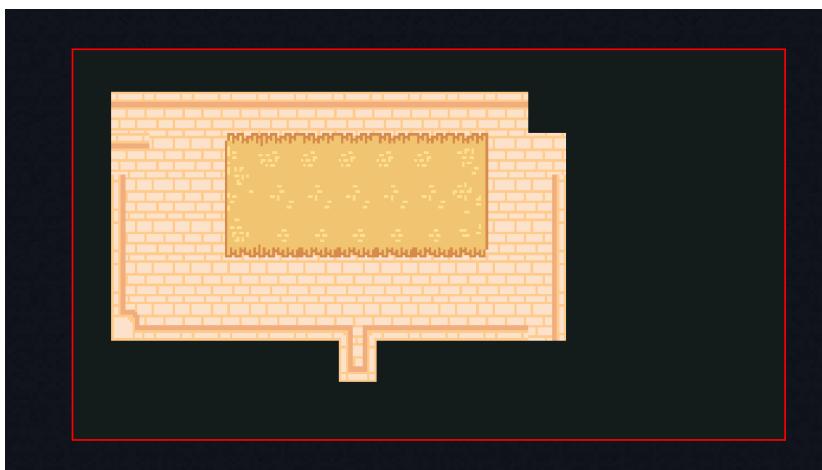
        //Lower Layer
        const lowerLayer = new Image(); //creates a new image
        //The source is the attribute of the instance
        lowerLayer.src = "/Maps/starting house/lower layer.png";
        //Calls the method for drawing the image
        this.drawMap(lowerLayer,0,0);

        //Collision Layer
        const collisionLayer = new Image();
        collisionLayer.src = "/Maps/starting house/collision layer.png";
        //this.drawMap(collisionLayer,0,0
        //Upper Layer
        const upperLayer = new Image();
        upperLayer.src = "/Maps/starting house/upper layer.png";
        //this.drawMap(upperLayer,0,0);

    }
}
```

I am only testing one of the images, as you can see in the code snippet above, I have commented out the drawMap() calls for the upper and collision layer, only drawing the lower layer at the default position of (0,0) coordinates. I am only testing the following line of code in this test iteration taken from the code snippet above: `this.drawMap(lowerLayer,0,0);`

1st Test: First Iteration Test Results



Test was successful.

The lower layer image has been drawn in the correct position with the right source and the `drawMap()` method seems to be working correctly

2nd Test: First Iteration

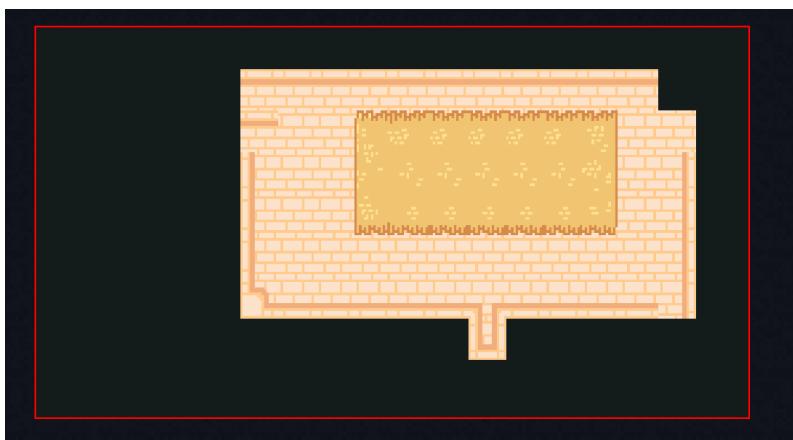
In this second test, I will be configuring the position coordinates of the images that the game will need to draw, this test will be separated into two different parts to ensure that the both PosX and PosY coordinates variable works effectively and correctly.

Testing the x position

I set the x coordinate position to 70, I am expecting the image to shift to the right by 70 units.

```
class Game{  
    //...  
    //function that draws images on the  
    drawMap(imageInstance, posX, posY) {  
        //The parameters is the image object, x and y position that will be  
        imageInstance.onload = () => {  
            posX = 70  
            posY = 0  
            this.context.drawImage(imageInstance, posX, posY)  
        }  
    }  
    //...  
}
```

2nd Test: First Iteration Test Results (Part 1)



The test was successful.

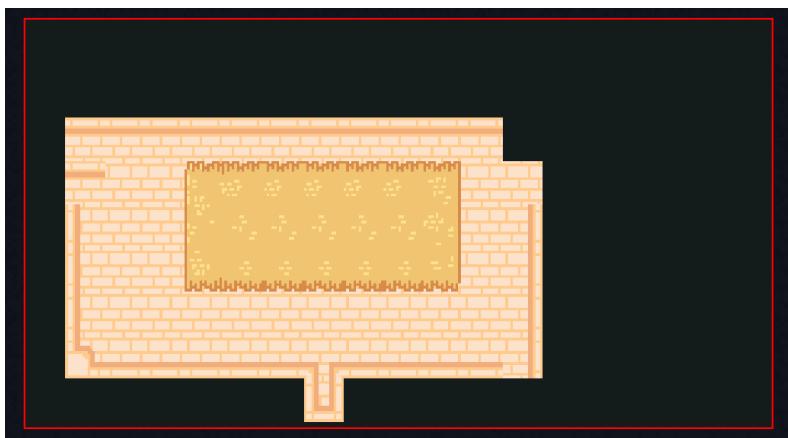
The lowerLayer image instance has been drawn in the correct position of (70, 0) coordinates, meaning that the image is drawn 70 pixels away from the left side of the canvas.

Testing the y position

I set the Y position to 20, I am expecting the image to shift down by 20 pixel units.

```
class Game{
//...
//function that draws images on the
drawMap(imageInstance, posX, posY) {
    //The parameters is the image object, x and y position that will be
    imageInstance.onload = () => {
        posX = 0
        posY = 20
        this.context.drawImage(imageInstance, posX, posY)
    }
}
}
```

2nd Test: First Iteration Test Results (Part 2)



The test was successful.

The lowerImage is shifted by roughly 20 pixel units to the south direction as expected, indicating that the position coordinates for drawing maps onto the canvas is easily configurable by using the PosX and PosY.

3rd Test: First Iteration

This test follows a boundary validation rule, meaning that I am checking if the game is able to handle any errors, if the **drawMap()** method is able to handle excess map outside the canvas, then the test will meet its criteria and be successful. The best and common way to handle this is to hide the excess sections of the map.

I have also broken down this test into 2 different sections to test the x and y axis

Testing on the x axis

```
//function that draws images on the
drawMap(imageInstance, posX, posY) {
    //The parameters is the image object, x and y position that will be
    imageInstance.onload = () => {
        posX = -100
        posY = 0
        this.context.drawImage(imageInstance, posX, posY)
    }
}
```

3rd Test: First Iteration Test Results Results (Part 1)



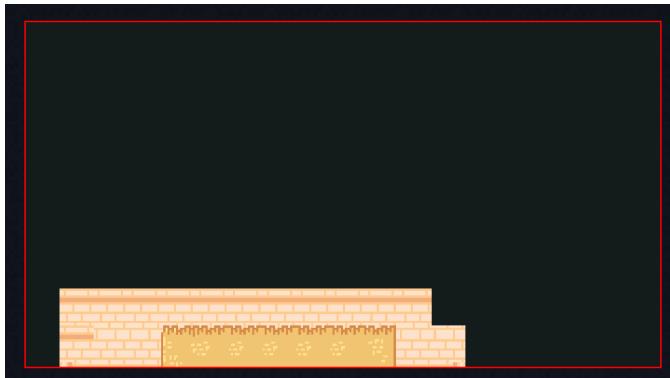
The test was successful,

If there are excess areas of an image outside the canvas on the x-axis, the game canvas manages to hide it. This test will allow me to develop other features such as map switching and character camera

Testing on the y axis

```
//function that draws images on the
drawMap(imageInstance, posX, posY) {
    //The parameters is the image object, x and y position that will be
    imageInstance.onload = () => {
        posX = 0
        posY = -100
        this.context.drawImage(imageInstance, posX, posY)
    }
}
```

3rd Test: First Iteration Test Results Results (Part 2)



The test was successful

If there are excess areas of an image outside the canvas on the y-axis, the game canvas manages to hide it. I have completely tested this feature and can proceed with the next tests stated in the test plan.

4th Test: First Iteration

I will be testing the functions which will draw the layers on the canvas.

```
//The init method will start the game
init() {

    //Lower Layer
    const lowerLayer = new Image(); //creates a new image
    //The source is the attribute of the instance
    lowerLayer.src = "/Maps/starting house/lower layer.png";
    //Calls the method for drawing the image
    this.drawMap(lowerLayer,0,0);

    //Collision Layer
    const collisionLayer = new Image();
    collisionLayer.src = "/Maps/starting house/collision layer.png";
    this.drawMap(collisionLayer,0,0);

    //Upper Layer
    const upperLayer = new Image();
    upperLayer.src = "/Maps/starting house/upper layer.png";
    this.drawMap(upperLayer,0,0);

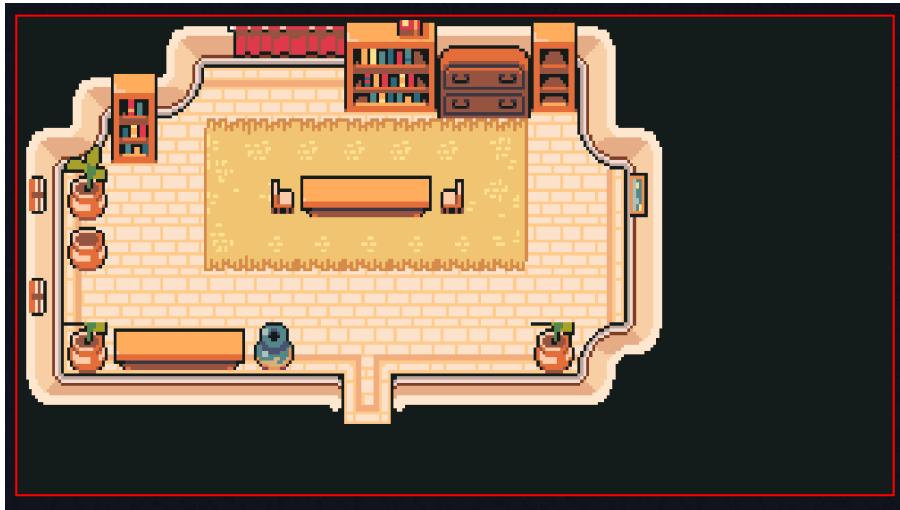
}
```

The requirements is to get the layers in the correct order:

1. Lower Layer
2. Collision Layer
3. Upper Layer

4th Test: First Iteration Test Results

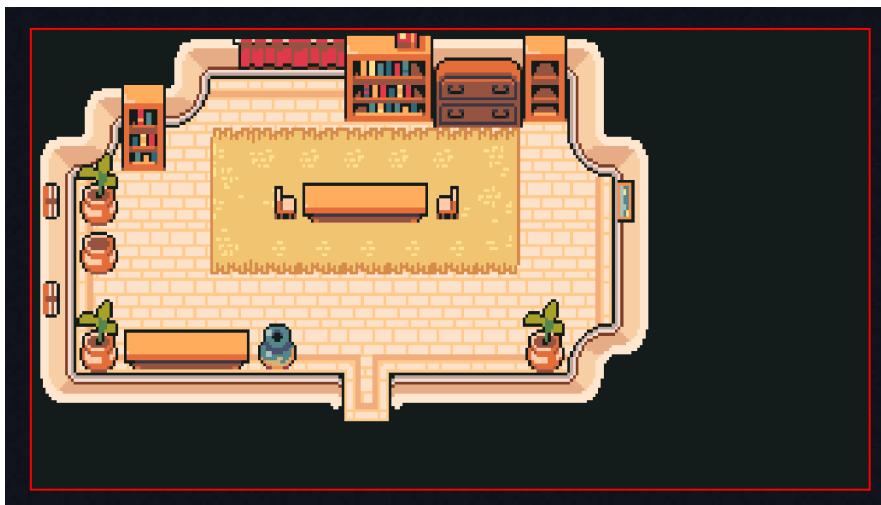
1st Result



2nd Result



3rd Result



Although the code stays constant, the results keep on changing every time I run the code. After researching the issue, I have concluded that the issue revolves around the browser itself, because the browser draws the image that has been loaded faster than the other images, and most images in this game will vary in different file size etc. Higher the size of the image, the longer it takes to load it, which could delay the largest layer to be drawn, and the results tend to be inconsistent as we have just seen from this iterative test.

Therefore, our results vary depending on the internet speed that it takes to load an image. One solution to this problem is to implement a game loop, which I was planning to do either way. However, I was planning to use the game loop for animation and character movement instead. Henceforth, I need to make changes to my original design and include map drawing within the game loop.

In summary, **The test was not successful.** To fix this issue, I will require the implementation of a game loop, which is the next feature I am going to implement.

Game Loop

```
//Game Loop
Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game Loop function
        requestAnimationFrame(() => {
            //Calls a function before going repainting the next frame

            gameLoop(); //Reiterates the function
        })
    }
    gameLoop(); //Initiates the game Loop
}
```

I have added a game loop method to the **Game class** in game.js, after researching for a suitable game loop for my game. I have declared the game loop function, which will continuously run and update the game. It works by repeating a set of instructions, typically the game logic and rendering of the game graphics, at a regular interval over and over again.

The **Loop()** method is the fundamental aspect of this game where it sets up and then runs the main game loop. This method defines and stores the **gameLoop()** which updates the game state and renders each frame. The **gameLoop()** function uses a '**requestAnimationFrame**' function which is a JavaScript method that allows a web page to request that the browser call a specified function to update a frame or animation before drawing the next set of images. It helps create smoother, more efficient animations and prevents unnecessary CPU usage.

There are alternative approaches to this such as using a **setInterval()** function, which may be easier to code and implement compared to a game loop system

I could use a **setInterval()** function instead, however, it's not suitable in the context of my game because of the lack of optimisation and potential for choppiness. Whereas the **requestAnimationFrame()** is better for performance and allows for smoother animation, as it is optimised to match the refresh rate of the user's device.

The **requestAnimationFrame()** function also calls the **gameLoop()** function after the next frame has been drawn onto the screen, this creates a recursive loop that runs non-stop. Finally, to initialise the **gameLoop()**, I have called the function again so it starts looping through the methods inside the **gameLoop()** continuously.

```
//The init method will start the game
init() {
    this.Loop();
}
```

I have deleted the previous code in the **init()** method, including calling the **drawMap()** to draw the layers. The **init()** method will initialise the game loop.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the game loop iterates itself	Game.init(), Game.Loop(), console.log()	Should print the same text
2	Check if the game loop runs more than one function	Game.init(), Game.Loop(), console.log()	Should print the first text than second text in the correct order until the code it stopped

The first test is a **normal** test which determines whether the game loop is iterating properly and it requires the **init()** method to call the **Loop()** method, I will include a `console.log()` function which will output a message, and if the **Loop()** works perfectly, the message will be output over and over again indicating that the game loop is active and repeating as expected.

The second test also follows a **normal** test to determine whether the game loop is capable of performing many functions in the correct order. To test this I am also using the 2 `console.log()` function with different outputs, the game loop should output both results consecutively and repeatedly until it is stopped to ensure that the game loop works as expected.

1st Test: First Iteration

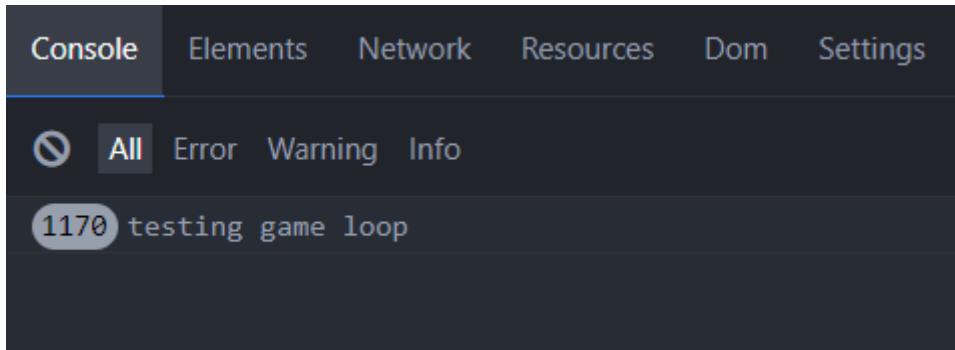
For the following test, I am expecting the game to display "testing game loop" repeatedly which will test if my game loop is working correctly as it is being looped over and over again. This will allow the game to render each frame and execute every function inside the loop in a correct order for the future development of different features.

```
//Game Loop
Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game Loop function
        requestAnimationFrame(() => {
            //Calls a function before going repainting the next frame

            console.log("testing game loop")

            gameLoop(); //Reiterates the function
        })
    }
    gameLoop(); //Initiates the game Loop
}
```

1st Test: First Iteration Test Results



The "testing game loop" is being displayed to the console each time a new frame is generated, proving that the game loop is working as intended.

The test was successful.

2nd Test: First Iteration

```
//Game Loop
Loop() { //Loop method for the game
  const gameLoop = () => { //Declares a game loop function
    requestAnimationFrame(() => {
      //Calls a function before going repainting the next frame

      console.log("testing game loop")
      console.log("testing game loop 2")

      gameLoop(); //Reiterates the function
    })
  }
  gameLoop(); //Initiates the game loop
}
```

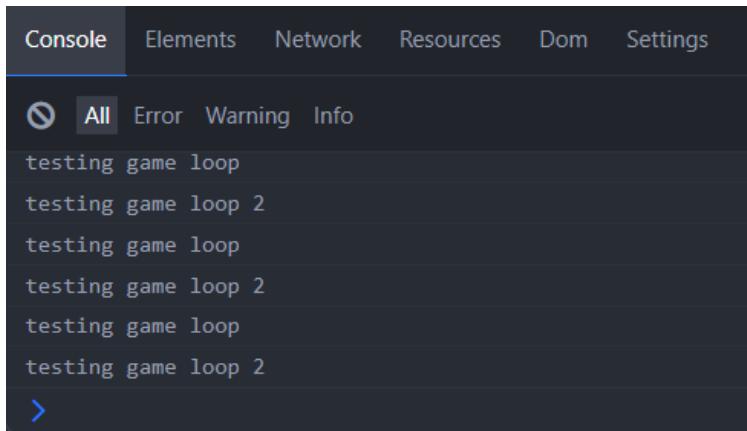
In the second test, I am testing the game loop function's ability to execute many statements simultaneously and in the proper order.

The loop should keep running and repeatedly output both messages to the console until the game is stopped or exited, thus it is also testing that the **requestAnimationFrame()** method and the recursive call to gameLoop() are working properly.

This test differs from the previous one in that it evaluates the functionality of several statements within the game loop, their correct execution sequence, and the game loop's continued proper operation with the additional additions.

In the code snippet, I have added two console.log() functions which will allow me to test if the game loop will run these two functions consecutively and repeatedly.

2nd Test: First Iteration Results



A screenshot of a browser's developer tools Console tab. The tab is titled 'Console' and has tabs for 'Elements', 'Network', 'Resources', 'Dom', and 'Settings'. Below the tabs, there is a filter bar with a 'All' button highlighted, and other buttons for 'Error', 'Warning', and 'Info'. The main area shows a list of log entries: 'testing game loop', 'testing game loop 2', 'testing game loop', 'testing game loop 2', 'testing game loop', 'testing game loop 2', and a blue arrow pointing right at the bottom.

The results are as expected, the game loop executes each function in itself in the correct order, and it is still being outputted repeatedly as it was tested in the first test.

The test was successful.

Map Class

Now that our canvas can draw images and layers, we can implement a constructor for Maps because we need multiple maps throughout the game and drawing three layers for each map will reduce our flexibility and lengthen our code unnecessarily. Therefore, it is more appropriate to create a new file called 'Map.js' and create a Map class:

Map Class - Attributes

```
class Map {
    constructor(config){

        this.name = config.name;

        //Creates Layer instances and assigns Layer sources

        //Lower Layer
        this.lowerLayer = new Image();
        this.lowerLayer.src = "/Maps/" + this.name + "/lower layer.png";

        //Collision Layer
        this.collisionLayer = new Image();
        this.collisionLayer.src = "/Maps/" + this.name + "/collision layer.png";

        //Upper Layer
        this.upperLayer = new Image();
        this.upperLayer.src = "/Maps/" + this.name + "/upper layer.png";
    }
}
```

Map.js

The code above is similar to the previous implementation of creating different map layers, however, I pursued a more efficient, flexible and better approach by setting a new attribute to the map class called '**name**' which will be the name of the map and the same as the map folder name for that map. This new addition will prevent me from repeating the same code for creating different layer instances for every map I make.

Now the sources for the layers are being set using a template literal.

Overall, the changes from the original design improve modularity, flexibility, allows for easier map creation and as map layer sources are dynamically set based on the map name.

Map Class - Methods

```
class Map{
    //...
    //Draw the Layer image instance onto the context
    drawLower(context){
        context.drawImage(this.lowerLayer, 0, 0)
    }

    drawCollision(context){
        context.drawImage(this.collisionLayer, 0, 0)
    }

    drawUpper(context){
        context.drawImage(this.upperLayer, 0, 0)
    }
}
```

I have added different methods for the Map class, which are taken from the previous map drawing implementation, however, I have changed the layer drawing methods to be more efficient and avoid the inconsistent loading error that we had in our last test. This update should fix our errors and draw our layers in a perfect order. This is because previously, we created image instances, set their sources and drew them all in the same function, this led to an error of layer ordering,

The new approach should solve this problem, the image layer instances are set in the attribute, therefore, when the layer drawing methods are called, the layers are already created and ready to be drawn.

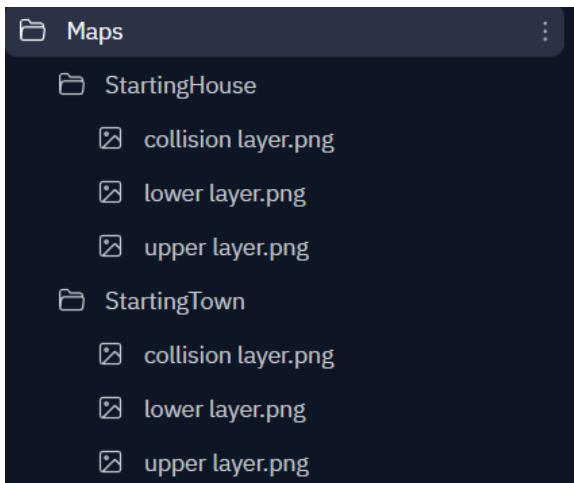
Alternatively, I could have used the `onload()` function to make sure the maps are loaded first before being drawn; however, my approach is better because it will facilitate map transition (when the player moves to a different map), because the map layers are set as attributes rather than temporary constants, so it would be difficult to set new sources to the map layers. Also, using the `onload()` function on each layer will unnecessarily lengthen our code, increase CPU usage, and make our code more inefficient.

Creating Map Instances

```
window.Maps = {
  StartingHouse: {
    name: "StartingHouse"
  },
  StartingTown: {
    name: "StartingTown"
  }
}
```

Globals.js

To create new map objects, I have decided to make a global object that contains all the different maps available. Currently I have two maps; “StartingHouse” and “StartingTown”.



Here's the folder format for each map. I have set the property for each map, which is the name used in their file path, this will let the code fetch their layer files easier as you can see in this figure.

I have created a new file called “**Globals.js**”,

where I will be storing all my objects that can be accessible from any module. This separation between object data and the code that uses it allows for better readability (especially as I am working on a project with lots of objects), it also encourages more modularity, easier debugging and easier to maintain.

```
window.mapDict = {
  StartingTown: new Map(window.Maps.StartingTown),
  StartingHouse: new Map(window.Maps.StartingHouse),
};
```

Globals.js

The following code creates a global dictionary object storing different instances of the map class. `window.mapDict` fetches map objects from `window.Maps` to create instances of the map class. This is done manually. However, I could go for an automatic approach to this by using an array of map objects, then loop through the array to create a dictionary of map instances.

I have coded an example of the alternate approach:

```
const maps = [
  { name: "StartingHouse},
  { name: "StartingTown"},
];

const mapDict = {};
maps.forEach((map) => {
  mapDict[map.name] = new Map(map);
});
```

Although this automated implementation of creating maps seems better, my initial approach, however, is more modular and easier to maintain because my implementation allows the map data to be easily shared between other files and modules. For example, I previously mentioned that I am going to add map switching. The Game class will require the map data that the player is currently using, therefore using window objects is more suitable for this project. Furthermore, my approach segregates map data from code that uses the same data, which opts for a more modular approach, and allows better readability and understanding of the code.

Linking to game class

```
// A game parent class which include every component of the game
class Game {
  constructor(config) {
    //HTML tag where it will display the game
    this.element = config.element;
    //Reference to the HTML canvas
    this.canvas = this.element.querySelector(".game canvas");
    //Will give us access to different drawing methods
    this.context = this.canvas.getContext("2d");
    //The current map that the user is playing on
    this.map = null;
  }
  //...
  init() {
    this.map = mapDict.StartingHouse;
    this.Loop();
  }
}
```

Game.js

Adding the map attribute to the game class will ensure easier switching between maps, as mentioned previously, every other class will be referenced and bridged in the game class. This.map holds the current map that the player is on right now. The **init()** method will initialise the game by setting the initial map and starting the game loop.

Drawing the maps

```

class Game{
    //...
    Loop() { //Loop method for the game
        const gameLoop = () => { //Declares a game Loop function
            //Calls a function before going repainting the next frame
            requestAnimationFrame(() => {

                //Drawing Layers
                this.map.drawLower(this.context);
                this.map.drawCollision(this.context);
                this.map.drawUpper(this.context);

                gameLoop(); //Reiterates the function
            })
        }
        gameLoop(); //Initiates the game Loop
    }
    //...
}

```

I am calling the **drawLayer()** methods in the **gameLoop()**, where it draws all the layers, and it is instantly being drawn through every loop cycle. This approach also facilitates map switching, because the game loop draws the current map that the player is on currently. So if the value of **this.map** is updated (meaning the player has switched to a different map), the game loop will now draw the layers of the new map rather than drawing the layers of the previous map. I could have not added the **this.map** attribute and passed in a parameter instead, however, this will require to set up a getter and setter method which will lengthen my code, cause inefficiency and unnecessary CPU usage.

I have added the map draw layers methods as separate functions rather than creating one function like the following:

```

Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game Loop function
        //Calls a function before going repainting the next frame
        requestAnimationFrame(() => {
            this.map.drawMap(this.context)
            //...
        })
        //...
    }
}

```

Test

Test No.	What I am testing	Test Code	Result Required
1	Check if the layer are ordered correctly	game.map.drawLowerlayer() game.map.drawCollisionLayer() game.map.drawUpperLayer()	The lowerLayer should be at the bottom, the upperLayer at the top and the collisionLayer in the middle.
2	Check if the game can add and select a different map with a different set of layers.	Game.map windows.Maps windows.MapDict	The game should be able to output and handle other maps with different layers, by a simple configuration to the code

This test plan emphasises on handling and drawing the map correctly.

The first test is a **normal** test which determines whether the layers are correctly sorted and ordered, the test code will call the three layer-drawing functions in the correct order as it is ordered in the game loop. This test is an extension from the previous test that failed the map layer ordering test in the 'Drawing Layers' section.

The second test is also a **normal** test which determines whether the game can handle and output a different map with a different set of layers. This should be easily configurable by only changing the **game.map attribute** and setting this attribute to the new map instance created using windows.Maps and windows.MapDict. This test is important because it will allow and facilitate the development of future features such as map switching and transition.

1st Test: First Iteration

In this test, I am reiterating the previous test from the ‘drawing layers’ section, where the game drew layers inconsistently. I need to implement my drawing functions in a game loop to fix this issue.

```
//...
requestAnimationFrame(() => {

    //Drawing Layers
    this.map.drawLower(this.context);
    this.map.drawCollision(this.context);
    this.map.drawUpper(this.context);

//...
})
```

Now that I have implemented this, this issue should be solved.

The following code highlighted in green should draw my layers accordingly to their order.

This test is an extension to the one of the previous test that was **unsuccessful**, which was the **4th Test: First Iteration** from the ‘Drawing Layers’ feature under the Game Class section , where the test was unsuccessful because the map layers were not being drawn with the correct order because of the lack of a game loop.

1st Test: First Iteration Results

As expected the result now is always consistent and the layers are drawn in their respective order.

The test was successful.



2nd Test: First Iteration

```
//Creates global Maps object
window.Maps = {
    //Configures data for each map object
    StartingHouse: {
        name: 'StartingHouse',
    },
    StartingTown: {
        name: 'StartingTown',
    }
}

//Creates instances of map with the data from
window.Maps
window.mapDict = {
    StartingHouse: new Map(window.Maps.StartingHouse),
    StartingTown: new Map(window.Maps.StartingTown),
};


```

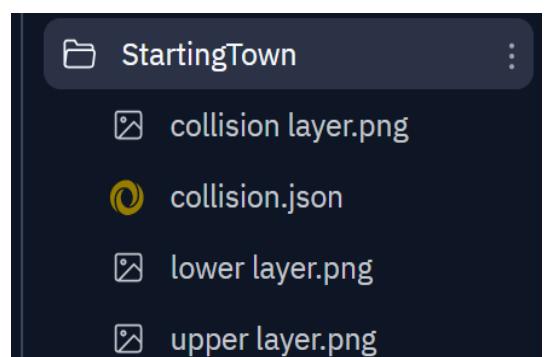
Global.js

In Global.js, I have added the data about the new map, by creating a new object, following the same structure and pattern of that of **startingHouse**, I have configured the attribute of the new map called “**StartingTown**” which is only the name of the map that the code is going to use for map’s file path for its layers.

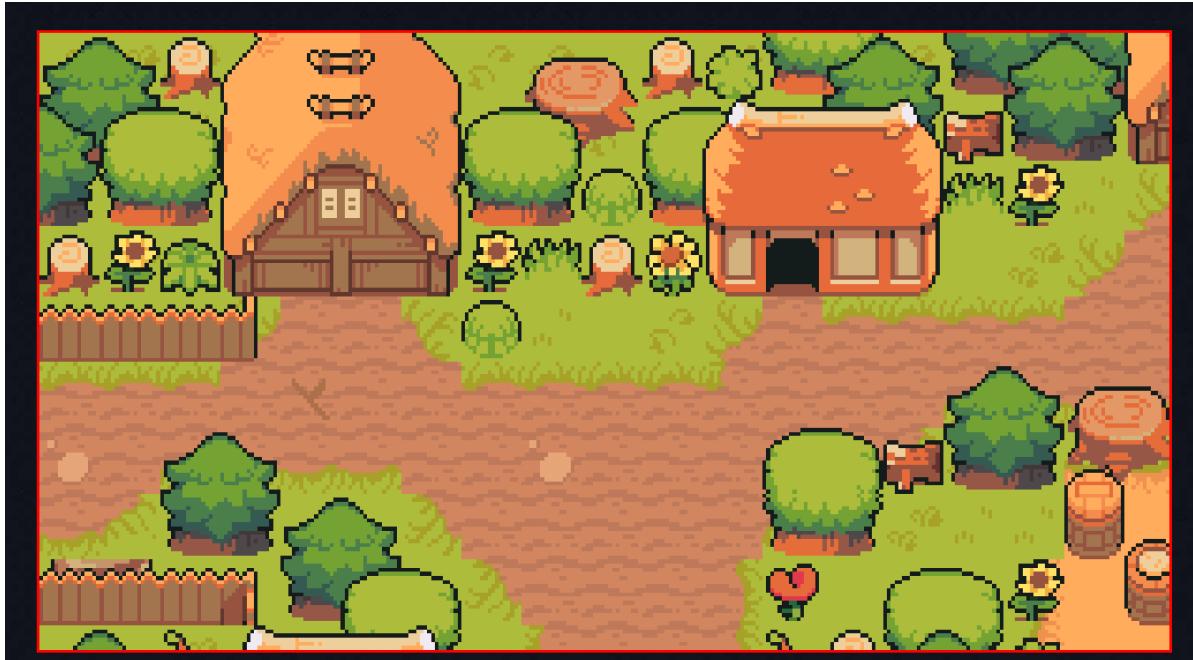
```
//...
init() {
    this.map = mapDict.StartingTown;
    this.Loop();
}
```

Game.js

To initialise this map, I have changed the initial map attribute of the Game class in the init() method. I imported the **StartingTown** map folder with the layers named with their correct titles.



2nd Test: First Iteration Results



After running the code, I expected the result to have been met as the game can now also draw different maps with the correct layers, this new feature will allow me to add several maps in the future.

The test was successful.

I have completed and met my first point in the success criteria

Success Criteria	Evidence	Justification
Multiple different maps	Screenshots of different maps	The first criteria is for the game to include many different maps in order to create an engaging world to explore for the user, as exploration is the main purpose and goal of my proposed solution. Instead, I could utilise a single map, which would be easier to manage and work with, decreasing development time. Adding numerous alternative maps, on the other hand, is a preferable solution because it allows for a more diverse gameplay experience, supplementing the game with a stronger sense of discovery.

Entities

The **Entity.js** file will include classes which are needed to create entities such as NPCs, monsters and the player. I started off by creating a new class called '**Sprite**'.

Sprite Class - Attributes

```
class Sprite{
    constructor(config){

        //Creates a new image attribute for the sprite
        this.skin = new Image();
        //Uses the name of the entity to find the source path
        this.skin.src = config.src
        //When the skin is loaded
        this.skin.onload = () => {
            this.isLoaded = true; //Mark the sprite as loaded
        }

        //Player movement animation
        this.animationsMap = config.animationsMap || {
            "idle-down" : [ [0,0] ],
            "idle-up" : [ [1,0] ],
            "idle-left" : [ [2,0] ],
            "idle-right" : [ [3,0] ],
            "walk-down" : [ [0,1], [0,2], [0,3], [0,0] ],
            "walk-up" : [ [1,1], [1,2], [1,3], [1,0] ],
            "walk-left" : [ [2,1], [2,2], [2,3], [2,0] ],
            "walk-right" : [ [3,1], [3,2], [3,3], [3,0] ]
        }

        //Obj Class
        this.Obj = config.Obj;
    }
}
```

Entity.js

This code defines a class called **Sprite** which represents a game character sprite. It has a constructor that takes a config object as an argument, which contains information about the sprite such as its name and animation frames.

The constructor creates a new image object skin for the sprite and sets its source path based on the config object. It also sets an onload event handler to mark the sprite as loaded once the image has finished loading.

I have added a new attribute called 'this.animationsMap' which will be useful later on for configuring animations for sprite, for example, character movement etc.

This attribute can be configured by taking the config object, however, if the config object is not provided for this attribute, the animationMap will be set to the predefined animation set. Either way, most entities and sprites in my game will have the same animation map, there are only some entities with different animation maps.

This attribute is a dictionary storing animation names as keys. Each animation is an array of coordinate pairs which represents the frames of the spritesheet to be used in the respective animation. The animation frames are stored as 2D arrays representing the row and column indices of the sprite sheet. For example, "idle-down" maps to a single frame, [[0,0]], while "walk-down" maps to an array of four frames, [[0,1], [0,2], [0,3], [0,0]].

The this.Obj attribute will store the actual object for this sprite, this attribute will store an instance of the Obj Class that I am going to use for the entity positions, behaviour, abilities etc. Whereas the sprite class is used for drawing related purposes for the entity such as drawing the entity, updating the animations and frames, setting the spritesheet etc.

Sprite Class - Methods

```
//Defines a draw method this entity
drawObj(context){
    //Fetches the positions from the Obj attribute
    const x = this.Obj.x;
    const y = this.Obj.y;
    //If the sprite is Loaded then draw the
    this.isLoaded && context.drawImage(this.skin,
        0, 0, 16, 16,
        x, y, 16, 16)
}
```

The Sprite class's drawObj method is defined in this code. The sprite picture is drawn using this technology in a certain context. The sprite's x and y locations are retrieved from the Obj property using the context as an input. The sprite is then drawn on the context at the designated position if it has been loaded. Sprite drawing could also be implemented by directly manipulating the canvas environment with low-level techniques like fillRect and drawImage. However, employing a sprite class of this kind enables more ordered and modular code, which may be simpler to read and maintain. Additionally, it makes it simple to configure sprite attributes like position and animations.

Object Class

```
// OBJECT CLASS
class Obj { //A blueprint for an object in the game
    constructor(config){
        this.x = config.x;
        this.y = config.y;
        //Creates a new instance of the sprite class
        this.sprite = new Sprite({Obj: this, src: config.src});
    }
}
```

Entity.js

In the new Obj class, I have added the main attributes which position and the sprite instance for this class. I could define each entity as simple JavaScript objects rather than OOP instances, as it is less complicated and easier to implement, however, this approach may not be efficient because it will require manually setting attributes and methods to each object.

Side Note: I have named the class ‘Obj’ instead of ‘Entity’ because I did not want to confuse the file name with the class name

Creating an instance of Obj

I have created the instance ‘player’ using the Object class, in the Global.js file:

```
window.Maps = {
    StartingHouse: { //Map Instance
        //Map Layer images
        lowerLayerSrc: "/Maps/starting house/lower layer.png",
        collisionLayerSrc: "/Maps/starting house/collision layer.png",
        upperLayerSrc: "/Maps/starting house/upper layer.png",
        entities: { //Stores every entity instance for this
            player: new Obj({ //Player instance is created
                x: 0, y: 0,
                src: "Characters/RedSamurai/SpriteSheet.png"})
        }
    }
}
```

Global.js

Each map object has an object called an entity property that contains a list of all the entities for that specific map. The player object is the sole entity declared for the "StartingHouse" map in the example you gave. The initial position (x and y) and source file for the player object's sprite are among the properties it has. The player object is defined as an instance of the Obj class (src).

```

class Map {
  constructor(config){
    //...
    this.entities = config.entities
  }
  //...
}

```

Map.js

Hence, I need to update the **Map class** in **Map.js**, by adding the **this.entities** property which is going to store all the entities in that map instance.

Overall, this approach of creating the '**entities**' attribute for the Map class is effective because it will allow me to configure which entity will be present in different maps. Additionally, since the Map class has access to the entities, in the future, when I add collisions for entities, they will have access to the map collision data. This creates a bridge between the map class and Obj class.

Drawing the entities

```

//Game Loop
Loop() { //Loop method for the game
  const gameLoop = () => { //Declares a game Loop function
    requestAnimationFrame(() => {
      //...
      //Stores all the entities in an array
      const entities = Object.values(this.map.entities);
      //Iterates through each entity
      entities.forEach(entity => {
        //Draws each entity
        entity.sprite.drawObj(this.context);
      })
      gameLoop(); //Reiterates the function
    })
  }
  gameLoop(); //Initiates the game Loop
}

```

I have updated the game loop to iterate through all the entities of the current map in order to draw them onto the canvas using the **this.sprite.drawObj()** method.

I have made a new constant variable called '**entities**', where I have stored an array of the values for an object's own enumerable properties that is returned by the **Object.values()** function. This list will allow me to iterate through each value in the array and call their draw methods by using the **forEach()** function for the corresponding entity to be drawn onto the canvas.

My approach is simple and effective, however, I could pursue a different approach by not creating a new variable to store my entities as a list and iterate through `Object.values(this.map.entities)` by itself such as in the following example:

```
Object.values(this.map.entities).forEach(entity => {
    entity.sprite.drawObj(this.context);
})
```

The main difference between these two codes is that my original approach creates a new variable for `Object.value(this.map.entities)` and iterates through this new variable. Whereas the alternative approach performs the `forEach()` immediately on the output of `Object.value(this.map.entities)`.

As a result, the alternate approach is more concise and reduces the length of the code. However, I choose my original approach because it is more readable and easier to understand as the objects I am iterating through are labelled correctly. Whereas the second code could potentially confuse the reader with the complex expression.

Furthermore, the '**entities**' variable could further simplify the code because I will be using this variable again in the game loop in the future, so rather than use

Object.variables(this.map.entities), I could reference it by only using '**entities**'.

Additionally, '**entities**' uses **Object.variables(this.map.entities)** function only once in a loop, whereas not using a variable will mean that the function is being called multiple times, which suggests that using a variable can provide performance benefits.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the entity is being drawn	Map.entities.Player, game.context	The entity should be visible on the canvas
2	Check if the entity position is correct	Map.entities.Player, player.x, player.y	When the coordinates position changes, the image position should correspond to the new coordinates.
3	Check if it draws entities with different sprite image	Map.entities.Player, player.sprite.src	The entity sprite image should be able to handle different images.
4	Check if it can display more than one entity	window.Maps.entities, game.context	There should be different and at least 2 entities visible in one map

This overall test plan, emphasises on the functionality of the entity class, its methods and the game's ability to handle different entities.

Test 1, is a **normal** test to determine whether the game is able to handle and draw an entity with the correct sizes which will allow me to verify if the game is capable of drawing an entity with the correct attributes and sprite sheet onto the canvas, I could have skipped this test or just could have checked this visually rather than programmatically, however, my approach of testing this class and methods will allow me to run the test numerous times and check that the game is drawn consistently and immune to changes in the system in the future development of my project.

Test 2 is also a **normal** test that determines the entity's position is appropriately monitored and updated, which will allow me to verify the game is capable of handling different positions for an entity on to the canvas, corresponding to the coordinates position of the entity. I could have easily avoided this testing phase by again visually verifying the entity's position, however, this test is important because automated tests are more reliable and consistent and can spot issues rapidly if the position of the entity is updated.

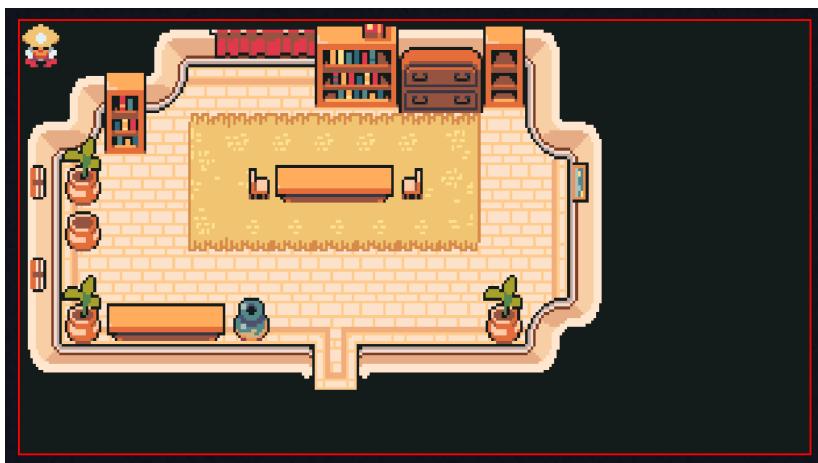
Test 3, is a normal test which determines whether the game is capable of displaying entities with different sprite attributes and sprite sheets, so I can make sure that all the sprites are correctly drawn on the sprite sheet and the game is able to handle more than just one sprite sheet. I could have not tested it and just checked this manually to see if the sprite picture changes for each thing, however, this test is important because the automated tests will allow me to verify if the code can handle different sprite images, especially when I am planning to add multiple entities with different sprite images, and this stage of development is the best time to check for any errors or bugs that may develop as a result of this functionality.

Test 4, the final test, is a normal test which determines whether the game is capable of handling multiple different entities onto one canvas which will help me verify the game's ability to load and draw many different entity sprites. I could have not tested this which could have made the process faster, however, using automated tests are a better approach of not testing at all because it will allow me to check if the code can handle different entities in one map effectively and display numerous objects, discover any difficulties and mistakes that may develop as a result of the development of these new classes and methods.

1st Test: First Iteration

In this test, I am checking if the entity is being drawn with the correct sprite and it should be visible on the canvas by only running the game and drawing the **player** instance from the entity class.

1st Test: First Iteration Test Result



The test was successful as the entity is displayed with the correct sprite and drawn onto the top right corner of the canvas, which is also the position (0,0).

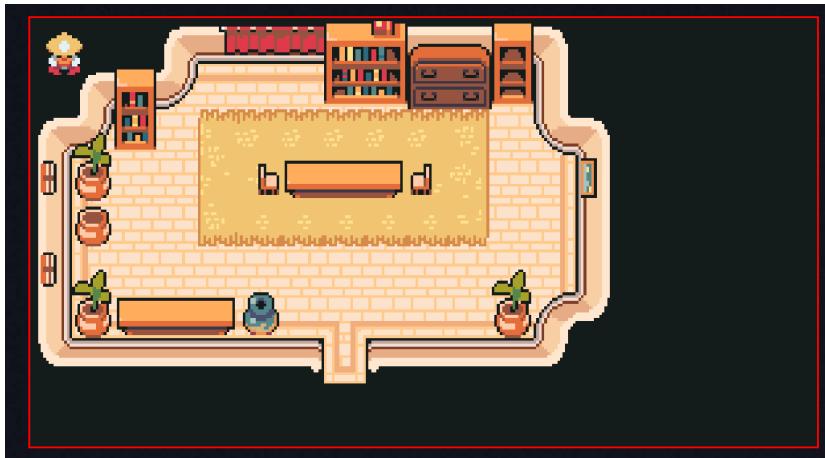
2nd Test: First Iteration

In the previous test, I set the 'player' position as (0,0) which seems to be correct as it is on the top left hand side of the canvas. In this test I need to try different x and y positions.

```
window.Maps = {
  StartingHouse: {
    //...
    entities: { //Stores every entity for this map
      player: new Obj({
        x: 5, y: 4,
        src: "Characters/RedSamurai/SpriteSheet.png"})
    }
  }
}
```

For this test, I have changed the x and y position values to 5 and 4 respectively.

2nd Test: First Iteration Test Result



The test did not go as expected, the player seems to have moved by a few pixels and it seems to be out of grid. The change is barely noticeable compared to the x and y position value of (0,0).

The test was unsuccessful.

Solution

To figure out an optimal solution for this error, I will be required to debug this issue appropriately. My approach to debugging this problem is to check whether the entity is inside a grid or not.

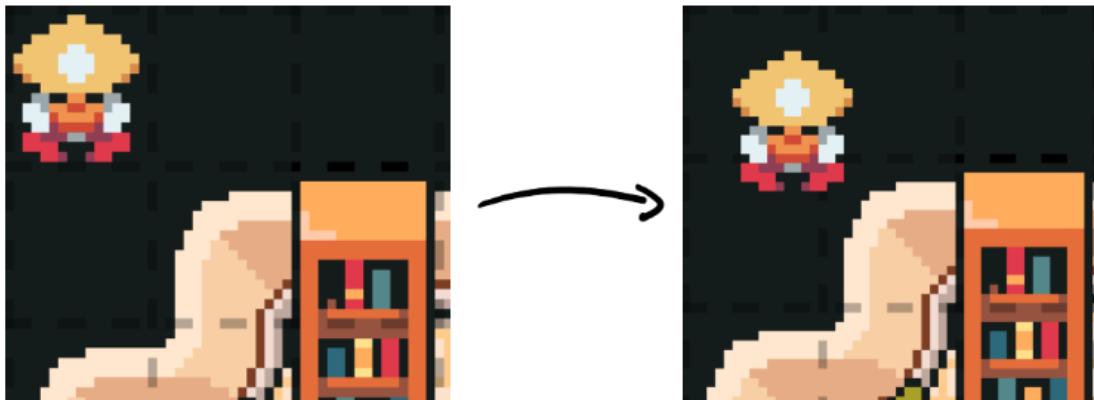
Debug #1

To solve this issue, I will need to add grids to the map so I can assess if there is any change when I configure the player position to a different value.



Here's the gridded map which will help me debug this issue.

I need to compare the results between when the player position was set to [0,0] to when it was set to [5,4]. Which will hopefully



```
//...  
player: new Obj({  
    x: 0, y: 0,  
//...  
})
```

```
//...  
player: new Obj({  
    x: 5, y: 4  
//...  
})
```

The results seem to differ, the player position is different by a few pixels, therefore we can conclude that the position does change, however it is not as expected, as I wanted the player to move accordingly to the grid, not according to the pixels.

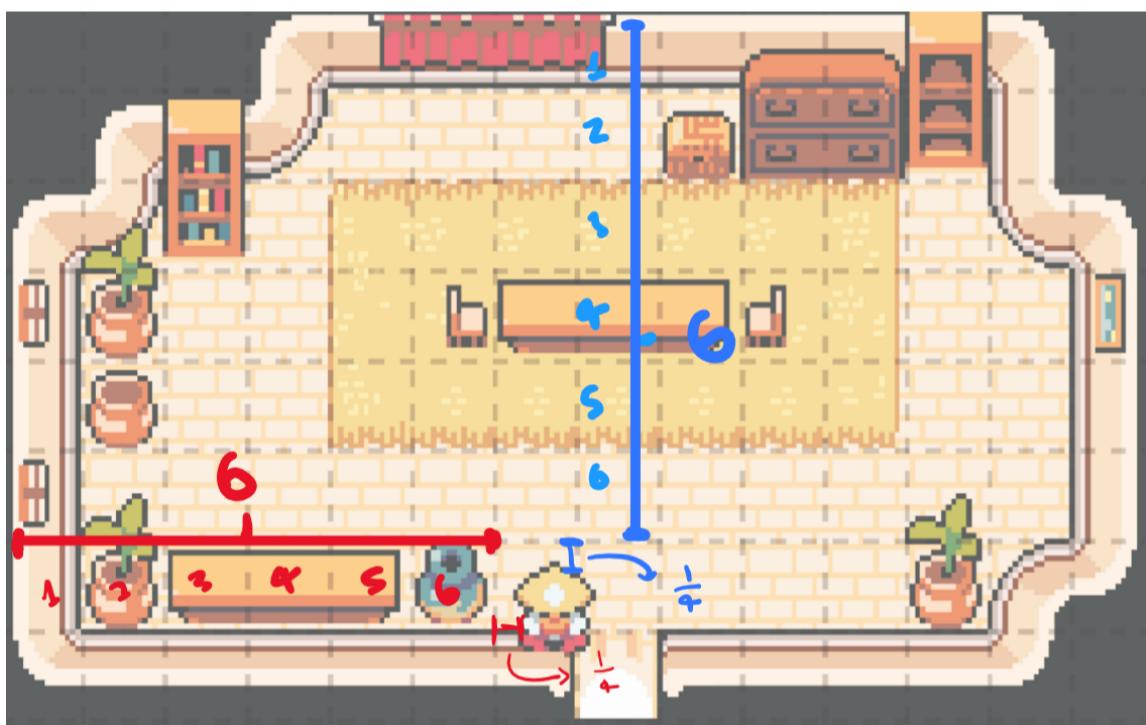
Meaning that x and y position of the Obj instances are in pixel value measurement, instead of grid tiles measurements.

Debug #2

```
//...
player : new Obj({
  x: 100, y: 100,
//...
```

In this second debug, I will be setting the x and y value to an extreme number such as '100' to examine further changes. My approach to this problem is to work out the size of one tile. To do so, I first need to work out how many grids it moves by using the gridded map.

After I obtain an estimate for how many tiles it is displaced by (the axis doesn't matter as the result is going to be the same on both axes) I can take the total pixel displacement, which is '100' in this case, and divide 100 by the estimated total displacement.



$$\begin{aligned} \text{Estimated tiles} \\ \text{Displacement} &= 6 + \frac{1}{4} \\ &= 6.25 \text{ files} \end{aligned}$$

$$\begin{aligned} \text{total pixel} \\ \text{Displacement} &= 100 \end{aligned}$$

$$\text{tile size} = \frac{100}{6.25} = 16 \text{ pixels} //$$

After the following calculation, I can conclude that the size of one tile is 16 pixels.

Solution:

I need to multiply the x and y position with 16 before drawing the entity.

2nd Test: Second Iteration

```
class Sprite{
    //...
    //Defines a draw method this entity
    drawObj(context){
        //Fetches the positions from the Obj attribute
        const x = this.Obj.x *16;
        const y = this.Obj.y *16;
        //If the sprite is Loaded then draw the entity
        this.isLoaded && context.drawImage(this.skin 0, 0, 16, 16, x, y, 16, 16)
    }
}
```

Entity.js

I have updated the sprite.drawObj() method where when the x and y coordinates are fetched from the Obj class attribute, the values are multiplied by 16, which is the height and width of one map tile. This should ensure that the entity is being drawn in the correct position.

```
window.Maps = {
    //...
    player: new Obj({
        x: 5, y: 4,
    //...
}
```

I am retesting the same x and y position set in the first iteration. The player should be displaced 5 tiles to the right and 4 tiles down.

2nd Test: Second Iteration Test Result



The player is positioned correctly as expected.

The test was successful.

3rd Test: First Iteration

```
window.Maps = {
  StartingHouse: {
    //...
    entities: { //Stores every entity for this map
      player: new Obj({
        x: 5, y: 4,
        src: "Characters/Boy/SpriteSheet.png"})
    }
  }
}
```

Global.js

In this test, I am testing if the game is able to handle a different sprite design. I have changed the source path of the player.

3rd Test: First Iteration Test Result



The test was successful.

However, there are some changes that I amend.

When switching to a different sprite, it would be easier to follow a similar implementation from the map class when it comes to fetching different character sprites.

In the map class, I have used a 'name' attribute which prevents me from repeating the same code for creating different layer instances for every map I make and the sources for the map layers are being set using a template literal. E.g. **“Map/”+map.name+”/layer.png”**

Since all of my characters follow the same file path format for their spritesheets, I can implement the same structure.

Refactoring

```

class Sprite{
    constructor(config){

        //Obj Class
        this.Obj = config.Obj;

        //Creates a new image attribute for the sprite
        this.skin = new Image();
        //Uses the name of the entity to find the source path
        this.skin.src = "Characters/" +this.Obj.name +"/SpriteSheet.png"
        //When the skin is Loaded
        this.skin.onload = () => {
            this.isLoaded = true; //Mark the sprite as Loaded
        }
        //...
    }
    //...
}

// OBJECT CLASS
class Obj { //A blueprint for an object in the game
    constructor(config){
        //Entity coordinates position
        this.x = config.x;
        this.y = config.y;
        //Name of entity sprite-sheet
        this.name = config.name;
        //Creates a new instance of the sprite class
        this.sprite = new Sprite({Obj: this, src: config.src});
    }
}

```

Entity.js

I have repositioned the **this.Obj** attribute of the sprite class to the top because I will need to use it to set the source of the sprite, as the **Obj class** now has a new attribute called **this.name**. Because of this new addition, the '**src: config.src**' is no longer needed for initialising the **this.sprite** attribute in the **Obj class** as **Obj.name** is enough to find the spritesheet of any character. I could stick to my previous approach, which will shorten my code, however, this approach makes it easier to create new entities, as I only need to pass in the 'name' attribute, then the code uses a 'template' to fetch their sprite sheet rather than the developer have to type whole source file path for every single entity they create. Additionally, it also decreases repetition as the file path source will not need to be repeated in windowMaps in Global.js when creating instances of **Obj**.

3rd Test: Second Iteration

```
window.Maps = {  
    //...  
    entities: { //Stores every entity for this map  
        player: new Obj({  
            x: 5, y: 4,  
            name: "Boy"})  
    }  
    //...  
}
```

Global.js

In the following test, the character sprite should change to “Character/Boy/SpriteSheet.png” by only taking in the name property: “Boy”.

3rd Test: Second Iteration Test Result



As you can see the results are the same, however, it's easier to change the character spritesheet solemnly by typing the name of the spritesheet rather than the whole file path

This approach is more effective and simpler, and I conclude that **the test was successful**.

4th Test: First Iteration

```
window.Maps = {
  //...
  entities: { //Stores every entity for this map
    player: new Obj({
      x: 5, y: 4,
      name: "Boy")},
    npc1: new Obj({
      x: 7, y: 4,
      name: "BlueSamurai")},
  }
  //...
}
```

Global.js

I have created a new entity called “**npc1**” following the same structure as the player. I’ve passed in the required attributes; x, y and the name property. The code should be able to handle multiple entities because the game loop iterates through each entity in the map and runs the draw function for each.

4th Test: First Iteration Test Result



The test was successful as expected, my game can handle multiple entities at once.

I have successfully completed all the tests and met the requirements for each test as stated in the test plan, this will allow me now develop further features such as the player class.

Review

Here's the review of the major changes of developing this entity feature, where I have added several new classes with distinct attributes and methods. This feature is an important stepping stone for developing a player class and further features such as collision detection etc. After testing these changes, new algorithms and classes, I have updated and added all the correct code under this section. I am happy with the results as I was able to maintain the modularity and simplicity of my code even though this is a complex feature which revolves around a few classes and other key aspects of game development.

Sprite class

```

class Sprite{
    constructor(config){
        //Creates a new image attribute for the sprite
        this.skin = new Image();
        //Uses the name of the entity to find the source path
        this.skin.src = config.src
        this.skin.onload = () => {//When the skin is Loaded
            this.isLoaded = true; //Mark the sprite as Loaded
        }
        //Player movement animation
        this.animationsMap = config.animationsMap || {
            "idle-down" : [ [0,0] ],
            "idle-up" : [ [1,0] ],
            "idle-left" : [ [2,0] ],
            "idle-right" : [ [3,0] ],
            "walk-down" : [ [0,1], [0,2], [0,3], [0,0] ],
            "walk-up" : [ [1,1], [1,2], [1,3], [1,0] ],
            "walk-left" : [ [2,1], [2,2], [2,3], [2,0] ],
            "walk-right" : [ [3,1], [3,2], [3,3], [3,0] ]
        }
        //Obj Class
        this.Obj = config.Obj;
    }
    drawObj(context){
        //Fetches the positions from the Obj attribute
        const x = this.Obj.x *16;
        const y = this.Obj.y *16;
        //If the sprite is loaded then draw the image with the correct
        //spritesheet positions
        this.isLoaded && context.drawImage(this.skin,
            0, 0, 16, 16,
            x, y, 16, 16)
    }
}

```

```
// OBJECT CLASS
class Obj { //A blueprint for an object in the game
    constructor(config){
        this.x = config.x;
        this.y = config.y;
        this.name = config.name
        //Creates a new instance of the sprite class
        this.sprite = new Sprite({Obj: this});
    }
}
```

Entity.js

```
window.Maps = {
    StartingHouse: { //Map Instance
        //Map Layer images
        lowerLayerSrc: "/Maps/starting house/lower layer.png",
        collisionLayerSrc: "/Maps/starting house/collision layer.png",
        upperLayerSrc: "/Maps/starting house/upper layer.png",
        entities: { //Stores every entity instance for this
            player: new Obj({ //Player instance is created
                x: 0, y: 0,
                src: "Characters/RedSamurai/SpriteSheet.png"})
        }
    }
}
```

Globals.js

```
//Game Loop
Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game loop function
        requestAnimationFrame(() => {
            //...
            //Stores all the entities in an array
            const entities = Object.values(this.map.entities);
            //Iterates through each entity
            entities.forEach(entity => {
                //Draws each entity
                entity.sprite.drawObj(this.context);
            })
            gameLoop(); //Reiterates the function
        })
    }
    gameLoop(); //Initiates the game loop
}
```

Game.js

Player Movement

In my game, I will need to create a class which inherits from the Obj class: the Player class. This player class will allow the user to control an entity such as grid movement, using special abilities, etc. It uses the same attributes as Obj, however I will need to add some different specific attributes for the player class to allow the user to control their character, especially in the movement behaviour. Contrast to other objects and entities, on a 'player' instance is possible to be controlled and moved around by the user, in order to begin the player movement behaviours, we will first need to set the player class, the constructor and the attributes.

Person Class - Attributes

```
class Person extends Obj{ //GameObj that can be controlled by the user
    constructor(config) {
        super(config); //Inherits methods and attributes from Obj

        this.tilesLeft = config.movementLeft || 0;
        //How many grids the player has left to travel

        this.directionDict = {
            "up" : ["y", -1], "down" : ["y", 1],
            "right": ["x", 1], "left" : ["x", -1]
        } //Assigns the axis and the value for the corresponding direction

        update(){
        }
    }
}
```

Player.js

As the **Person class** extends the **Obj class**, it will acquire all of the parent class's attributes and methods. This is crucial since it enables code reuse and prevents us from using the same feature twice, which would have made it inefficient and lengthy.

The Person class will share the same fundamental characteristics as the Obj class, such as x and y coordinates and the sprite object. We may reuse the same code to draw the player on the canvas as a result.

I could create a new class rather than inheriting from the existing class. However, inheriting the Obj class will decrease the length of the code, and it is a more effective approach in OOP, especially when we want to increase reusability and extend the functionality of an existing class.

Tiled Based Movement

Tiled based movement is a type of movement system often used in game development that restricts character movement to a grid-based map. Characters can only move along the lines or nodes of the grid in a specific direction: up, down, left, or right. This type of movement system is often used in classic role-playing games, turn-based strategy games, and other genres.

I could use free-form movement, which allows the player to move their character in any direction with no restrictions. This type of movement system is often used in open world games. However, my game is a RPG style game, where tiled based movement is more suitable. Secondly, with tiled based movement, the coordinates will always be whole integers, this allows me to add cutscenes as I can predict which tile the player will move on more accurately.

Person Class - Methods

```
//The update method will commit changes to the player
update() {
    this.updatePos();
}
updatePos() {
    //If the player has any tiles left to travel
    if (this.tilesLeft > 0) {
        //Checks which direction it needs to move to
        const [axis, value] = this.directionDict[this.direction]
        //Changes their position value on the correct axis
        this[axis] += value;
        //The method will stop when the movementLeft is 0
        this.tilesLeft -= 1
    }
}
```

Player.js

The **update()** method is used to update the player's state, and it only calls the **updatePos()** method in this implementation, currently. The **updatePos()** method is in charge of handling the player's movement on the game map. Based on the **this.tilesLeft** attribute, it determines whether the player needs to move. If the player needs to move, it retrieves the direction from the **directionDict** object and moves them by updating their position value on the appropriate axis. The **this.tilesLeft** attribute is then decremented. Instead of having a separate **updatePos()** method, I could use an alternate design to integrate the movement logic directly in the **update()** method. The **this.tilesLeft** attribute and the **this.directionDict** object would be removed, but the **update()** method would become more complex and difficult to read.

Test

Test No.	What I am testing	Input	Result Required
1	Check if the player moves horizontally	This.direction, this.movementLeft	The character should be able to travel across the map horizontally

This test is a **normal** test which determines if the player is able to move one direction aided by the changes I have added to the code, this will help me verify if the entities are able to travel appropriately and reach their destination following the tile based grid pattern. I could skip this test, however, This test is very important, especially for future development because it will help me to develop more systems for movements such as controlled movement, animation movement, NPC behaviour loops etc. To show if the test was successful, I am going to use video evidence to test this, as it cannot be shown by simple console log messages or screenshots.

1st Test: First Iteration

I have configured the '**direction**' and '**TileLeft**' attribute. I am expecting the character to move towards the right direction by 4 tiles.

```
window.Maps = {
    StartingHouse {
        //...
        entities: { //Stores every entity for this map
            player: new Player({
                x: 5, y: 4,
                name: "Boy",
                direction: "right", TilesLeft: 4})
        }
    }
}
```

Globals.js

1st Test: First Iteration Results



The test has failed, the character did move to the right direction but it has not travelled 4 tiles, but less than 1 tile. One of the reasons is because each tile has 16 pixels, as we saw previously. The code reads 'TilesLeft: 4' as there are 4 pixels left to travel.

The test was unsuccessful

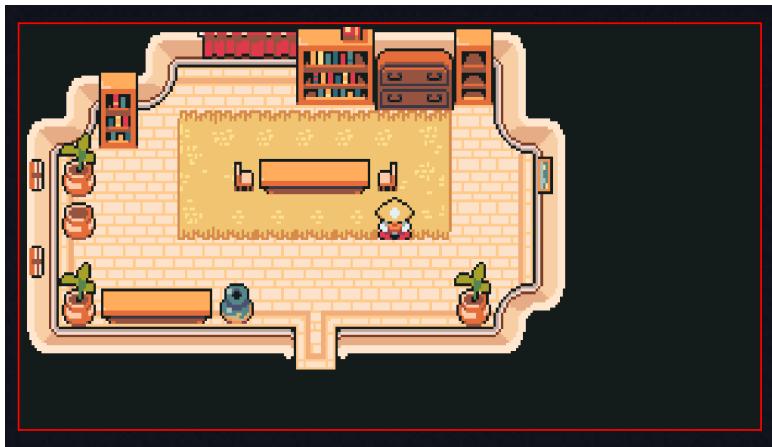
1st Test: Second Iteration

```
class Person extends Obj{ //GameObj that can be controlled by the user
    constructor(config) {
        super(config); //Inherits methods and attributes from Obj

        //How many grids the player has left to travel
        this.tilesLeft = config.movementLeft*16 || 0;
        //...
    }
}
```

I have updated the **this.tilesLeft** multiplied it by 16 to fix the issue, similarly to the entity movement that we saw previously, we need to make sure that the **this.tilesLeft** attribute is in the terms of tiles instead of pixels, and again, the size of each tile is 16x16 pixels, hence why I need to multiply the value by 16.

1st Test: Second Iteration Results



The player moves correctly in the right direction by 4 tiles.

Here's a video test:

[1st Test: Second Iteration Test Video](#)

The test was successful.

Controlled Movement

Created a new file called '**KeyInput.js**' where it would handle all the character keys inputted by the user.

keyInput class - attributes

```
class keyInput{
    constructor(){

        // Initialize an empty array to track held keys
        this.keysHeld = [];

        // Define a map that associates key codes with directions
        this.keyDirectionMap = {
            "ArrowUp": "up",
            "ArrowDown": "down",
            "ArrowLeft": "left",
            "ArrowRight": "right",

            "KeyW": "up",
            "KeyS": "down",
            "KeyA": "left",
            "KeyD": "right"
        };
    }
}
```

KeyInput.js

I created a new class called '**keyInput**', where I create an array which will store keys that are held currently. I have defined a **keyDirectionMap** object that maps specific key codes to directional inputs. The directional inputs are based on the arrow keys and WASD keys, which are commonly used in video games for movement.

I could have pursued a different approach which is to allow the game to handle key inputs directly using the '**addEventListener**'. This approach could be easier as it won't require an entire class for the key inputs in the game, however, this approach will be more inefficient, decrease modularity, make the game code more unnecessarily complex, and my original approach allows for better structure and abstraction of the key input handling.

The main key input logic is implemented in a distinct class and can be easily adjusted or extended without affecting other parts of the code, which as I mentioned before it increases modularity and it is an effective approach of organising the code.

keyInput class - methods

```

class keyInput {
    //...
    init(){
        //Triggers when the user presses and releases a key, respectively
        document.addEventListener('keydown', this.handleKey.bind(this));
        document.addEventListener('keyup', this.handleKey.bind(this));
    }

    handleKey(event){
        //Gets the direction value from input key
        const direction = this.keyDirectionMap[event.code];
        //Gets the index of the direction in the keysHeld array
        const index = this.keysHeld.indexOf(direction);

        //If the held key is not in the keysHeld array...
        if (event.type === 'keydown' && !this.keysHeld.includes(direction)) {
            //...Then add the direction to the front of the keysHeld array
            this.keysHeld.unshift(direction);
        //If the released key is in the keysHeld array...
        } else if (event.type === 'keyup' && index > -1) {
            //...Then remove the direction of the corresponding key
            this.keysHeld.splice(index, 1);
        }
    }

    get direction(){
        return this.keysHeld[0];
    }
}

```

KeyInput.js

The **this.init()** method configures event listeners for the keydown and keyup events, which are generated when a user presses or releases a key on the keyboard. `bind(this)` is used to bind these events to the `handleKey` method, which guarantees that `this` keyword relates to the current instance of the `KeyInput` class.

The **this.handleKey()** method is in charge of determining the key press or release direction and updating the **this.keysHeld** array accordingly. The **this.keyDirectionMap** property maps key codes to direction values, and the `indexOf()` method determines whether a direction already exists in the **this.keysHeld** array.

Since it allows for responsive and easy controls, this method of processing key input is a common trend in JavaScript game development.

I could use a library like `keypress.js` or `p5.js`, which provide additional functionality for managing keyboard and mouse events. However, manually integrating key handling logic using event listeners provides a lightweight and flexible option that can be easily adjusted to meet individual needs.

The `keyInput` class's direction getter method returns the first element of the `this.keysHeld` array, which reflects the current direction being held down by the user. This is a quick and easy way to get the user's current direction without having to cycle over the entire `keysHeld` array or save it in a separate variable. Alternatively, I could use a boolean flag for each direction input indicating whether it is currently held down or not, but this would require more code and make handling several directions pressed simultaneously more difficult.

```
class Game {
    //...
    Loop(){
        //...
        //Draws every single entity
        entities.forEach(entity => {
            entity.update({
                arrow: this.directions.direction
            })
            entity.sprite.drawObj(this.context);
        })
        //...
    }

    //The init method will start the game
    init() {
        this.map = new Map(window.Maps.StartingHouse)
        this.directions = new keyInput()
        this.directions.init()
        this.Loop();
    }
}
```

Game.js

With this code, I've added an `entity.update()` function to the game loop, which takes an object as an argument and has an `arrow` property. On each iteration of the game loop, this method call is made on each entity object in the `entities` array.

The `update()` method is used to change the internal state of each entity object in response to user input. The `arrow` attribute, in particular, is utilised to specify the direction of travel for each entity.

Each entity object can handle their own state and behaviour using this approach, rather than relying on a centralised system. This can result in code that is more modular and maintainable, as well as more flexible and changeable behaviour for each entity.

I could add a centralised input system that will update the state of each individual entity. However, this approach could be inefficient and lengthen my code especially as the number of entity objects will increase later on, which would make it harder to maintain.

```
class Player{
    //...
    //The update method will commit changes to the player
    update(state) {
        this.updatePos();

        //If the player has reached no tiles left
        //and direction key is pressed
        if (this.TilesLeft === 0 && state.arrow){
            //Update the direction attribute to the new direction input
            this.direction = state.arrow;
            //The player needs to travel 1 tile.
            this.TilesLeft = 1*16;
        }
    }
}
```

Player.js

Now we need to use this state that is being passed on. We take the state object as a parameter for **this.update()** in the player class. If the player has no tiles remaining to move and a direction key is hit, the if statement in the player class's update method checks. If this condition is met, the player's direction attribute is updated to reflect the new direction input, and the number of tiles left to move is set to 16.

Test

Test No.	What I am testing	Test Code	Result Required
1	Check if eventListeners updates the keysHeld Array.	keyInput.handleKeyEvent()	<p>When a key is pressed, the corresponding direction should be placed in front of the array.</p> <p>When a key is released, the corresponding direction should be removed from the array</p> <p>Other keys should be ignored and not taken</p>
2	Check if the character movement works	player.update() this.directions.init()	The character should move subject to the keys inputted. E.g. If the right arrow key is pressed once, the character should move to the right only by one tile

The first test is a **boundary** test, which determines if the game can handle both valid and invalid key inputs which will ensure if the array is working correctly and updated with the proper values. The main purpose of this test is to check whether the **keyInput** class is able handle different key inputs, update the **this.keysHeld** array accordingly to the inputs and take account whether the inputs are valid or not when updating these attributes.

I could not carry out this test, however, It is important to test if eventListeners updates the **this.keysHeld** array as it relies on the keyboard inputs in order to execute player movement. If the array is not appropriately updated, the character movement may not function and output the unexpected results, which will negatively affect user experience.

The second test is a **normal** test which determines whether the Player class's **update()** method allows the player to move with the correct distance, direction and other key attributes. The player should travel one tile at a time in the direction of the last key input in the keysHeld array in order to meet the requirements and success criteria for this test.

I could not test for this and just carry on with my development process, however, It is important to test character movement because it helps me verify if the character moves to the correct direction corresponding to their respective key input, for example, if the W key or Up arrow key is pressed, the entity should move north. If this feature does not output the right expected results, this could cause dissatisfaction and confusion among the users.

1st Test: First Iteration

```

handleKey(event){
    //Gets the direction value from input key
    const direction = this.keyDirectionMap[event.code];
    //Gets the index of the direction in the keysHeld array
    const index = this.keysHeld.indexOf(direction);

    //If the held key is not in the keysHeld array...
    if (event.type === 'keydown' && !this.keysHeld.includes(direction)) {
        //...Then add the direction to the front of the keysHeld array
        this.keysHeld.unshift(direction);
    //If the released key is in the keysHeld array...
    } else if (event.type === 'keyup' && index > -1) {
        //...Then remove the direction of the corresponding key
        this.keysHeld.splice(index, 1);
    }
    console.log(event.code, this.keysHeld)
}

```

The `console.log` statement, which I have added at the end of the `handleKey()` method, is used for testing and troubleshooting. It displays the latest key pressed and the current direction in the **keysHeld** array, which keeps track of the keys that are currently pressed. This will allow me to confirm that the array is being updated correctly as keys are pushed and released by logging the array.

This aids with the identification of any faults or bugs in the code that may influence the player's mobility or the game's operation. Overall, logging statements can help me monitor the behaviour of the **handleKey()** method code, identify any issues that need to be addressed and check if the **keysHeld** attribute is working correctly.

1st Test: First Iteration Test Results

```

KeyW ▶ ['up']
KeyW ▶ []
KeyD ▶ ['right']
KeyD ▶ []
KeyA ▶ ['left']
KeyA ▶ []
KeyS ▶ ['down']
KeyS ▶ []
ArrowLeft ▶ ['left']
ArrowLeft ▶ []
ArrowUp ▶ ['up']
ArrowUp ▶ []
ArrowRight ▶ ['right']
ArrowRight ▶ []
ArrowDown ▶ ['down']
ArrowDown ▶ []
KeyF ▶ [undefined]
KeyF ▶ []
KeyF ▶ [undefined]
KeyF ▶ []

```

The test wasn't entirely successful, although when the keys are held and released, the **keysHeld** attribute is updated. However, if an undefined key is pressed, the `handleEventKey` method enters it into the array.

The test was partially successful

1st Test: Second Iteration

```

handleKey(event){
    //Gets the direction value from input key
    const direction = this.keyDirectionMap[event.code];
    //Gets the index of the direction in the keysHeld array
    const index = this.keysHeld.indexOf(direction);

    if(direction){
        //If the held key is not in the keysHeld array...
        if (event.type === 'keydown' && !this.keysHeld.includes(direction)) {
            //...Then add the direction to the front of the keysHeld array
            this.keysHeld.unshift(direction);
        }
        //If the released key is in the keysHeld array...
        } else if (event.type === 'keyup' && index > -1) {
            //...Then remove the direction of the corresponding key
            this.keysHeld.splice(index, 1);
        }
        console.log(this.keysHeld)
    }
    console.log(event.code)
}

```

I have wrapped the event handler's function inside an if statement which will only run if direction is defined. I have also updated the debug code by splitting the console.logs functions.

```

22 KeyF
▶ ['right']

KeyD
▶ []

KeyD
▶ ['down']

ArrowDown
▶ []

ArrowDown

6 KeyL
▶ ['up']

KeyW
▶ []

KeyW

MetaLeft
ShiftLeft

```

1st Test: Second Iteration Test Results

The test was successful and the issue was resolved;

The array now only updates if a key is defined, otherwise the function will not run.

2nd Test: First Iteration

The following test should check if the player is able to move respectively to the direction input as the state is now passed on to the player update method.

```
handleKey(event){
    //...
    if(direction){
        //If the held key is not in the keysHeld array...
        if (event.type === 'keydown' && !this.keysHeld.includes(direction)) {
            //...Then add the direction to the front of the keysHeld array
            this.keysHeld.unshift(direction);
            console.log(this.keysHeld)
        }
    }
}
```

KeyInput.js

When a new direction key is added to the front of the **keysHeld** array, the `console.log()` function is used to output the current value of **this.keysHeld** to the browser console. This allows me to check the current status of the `keysHeld` array and confirm that key inputs are successfully registered.

2nd Test: First Iteration Test Results

[2nd Test: First Iteration video:](#)

The test was successful and as expected. - As seen in the test, the player is able to move around corresponding to the input keys shown in the **this.keysHeld** attribute in the console.

I have also met the following success criteria

Success Criteria	Evidence	Justification
Character Movement	Video of the character moving with the right direction input from the user.	The second criteria is to allow the user to have a smooth and responsive character movement, ensuring that the game is enjoyable for the user. I could restrict the character movement allowing only to travel in two directions or less, however, adding a 4 way character movement system makes my game more immersive, dynamic and even more enjoyable.

Speed Boost

Speed configuration

As previously mentioned, I am going to add a speed boost feature for the player. In order to do so, I will need to be able to configure the speed of the character movement, which will allow me to set custom speed to each entity and objects in the game.

```
class Obj { //A blueprint for an object in the game
  constructor(config){
    this.x = config.x*16;
    this.y = config.y*16;

    //Creates an attribute for the sprite or skin of the game object
    this.sprite = new Sprite({Obj: this, src: config.src});

    //The direction that the object faces
    this.direction = config.direction || "down";

    //Character movement speed
    this.speed = config.speed || 1
  }
}
```

Entity.js

I have added a new attribute in the **Obj class**, called **this.speed** where I configured the speed to be set whatever is in the config object. If **config.speed = 3**, for example, it suggests that the entity will move three times as fast as their walking speed, which is **this.speed = 1**. This approach is flexible as it will let me configure the speed movement of different object instances of the game.

I could add this **this.speed** attribute to **Player class** in **Player.js**, however this will remove the flexibility of adding custom speed to other entities except from the player, whereas in my approach I can add custom speed to entities and also to the player as the player class inherits the attributes from the Obj class.

```

class Player{
    //...
    updatePos() {
        if (this.TilesLeft > 0) { //If the player has to move
            //Checks which direction it needs to move to
            const [axis, value] = this.directionDict[this.direction]
            //Changes their position value on the correct axis
            this[axis] += value*this.speed
            //The method will stop when the TilesLeft is 0
            this.TilesLeft -= this.speed
        }
    }
}

```

Player.js

I have also updated the **updatePos()** method from the Player class in Player.js.

At this current stage, I am only going to add the speed configuration to the player, I can easily add speed configuration to the other entities later on.

In this following code, the player speed is multiplied by the value given the axis is changed by when a movement occurs. Then I subtract **this.TilesLeft** by **this.speed**, if the **this.speed** is 2, the player position and **this.TilesLeft** is going to decrease twice as fast, meaning that the player will travel twice as fast than walking.

Speed Configuration Tests

Test No.	What I am testing	Test code	Result Required
1	Check if the character speed changes	this.speed,	When there is a change to the speed attribute, the character should move at a different speed in respect to ' this.speed '

This test is a normal test where I am checking if the speed of the entity changes when configured to a different speed. How fast the player is travelling should directly correspond to the **this.speed** value. I am only executing one test for this new feature because there could possibly be a few errors with the new changes because I am not confident with the robustness of these new changes. If this single test is successful, I can move on to the next stage, where the user will be able to toggle between 'running' and 'walking' using the speed configuration feature.

1st Test: First Iteration

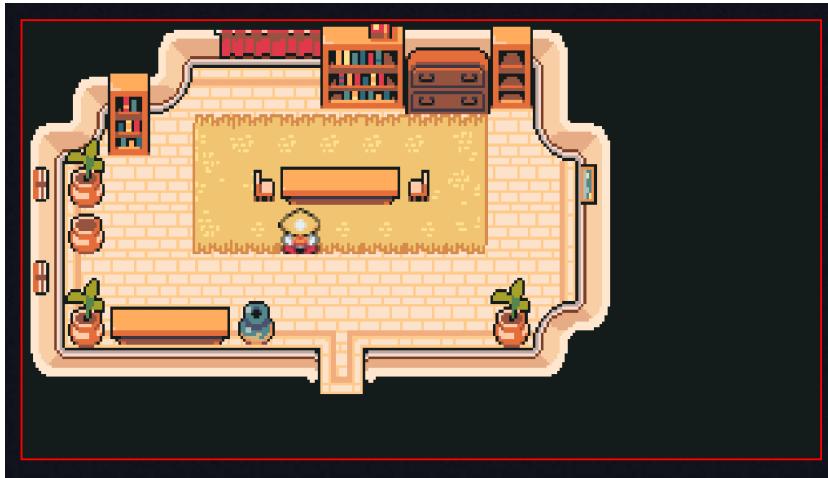
```
window.Maps = {  
  StartingHouse {  
    //...  
    entities: { //Stores every entity for this map  
      player: new Player({  
        x: 5, y: 4,  
        name: "RedSamurai",  
        speed: 1.5,  
      })  
    }  
  }  
}
```

Global.js

Thanks to the config object, I can configure the speed of the player in the window.Maps global object. I have set speed to 1.5 to test for speed change.

1st Test: First Iteration Test Results

1st Test: First Iteration Video



The test was unsuccessful.

Issue:

There seems to be an issue with the player, when the player moves at this.speed = 1.5, the player seems to freeze.

Debug #1

```
class Player{
    updatePos() {
        //If the player has any tiles left to travel
        if (this.tilesLeft > 0) {
            //Checks which direction it needs to move to
            const [axis, value] = this.directionDict[this.direction]
            //Changes their position value on the correct axis
            this[axis] += value*this.speed;
            //The method will stop when the movementLeft is 0
            this.tilesLeft -= this.speed
            console.log(this.tilesLeft)
        }
    }
}
```

The `console.log()` function will help me figure out the changes to `this.tilesLeft` when the player travels.

Debug Test Results #1

14.5
13
11.5
10
8.5
7
5.5
4
2.5
1
-0.5
>

The reason for the player freezing issue is that `this.tilesLeft` becomes negative. In the current code, there is no limit for the player to stop when `this.tilesLeft` reaches below zero when the `this.tilesLeft` attribute is being subtracted a number greater than 1.

Solution:

When the `this.tilesLeft` attribute reaches to 0 after being subtracted from `this.speed`, the `this.tilesLeft` should be back to 0, this prevents `this.tilesLeft` from becoming a negative number and the player from freezing.

1st Test: Second Iteration

```

class Player{
    //...
    updatePos() {
        //If the player has any tiles left to travel
        if (this.tilesLeft > 0) {
            //Checks which direction it needs to move to
            const [axis, value] = this.directionDict[this.direction]
            //Changes their position value on the correct axis
            this[axis] += value*this.speed;
            //The method will stop when the movementLeft is 0
            this.tilesLeft -= this.speed
            if(this.tilesLeft < 0){
                this.tilesLeft = 0
            }
            console.log(this.tilesLeft)
        }
    }
}

```

Now, If this.tilesLeft reaches negative, set it back to positive, theoretically, it should prevent the player from freezing and unable to move.

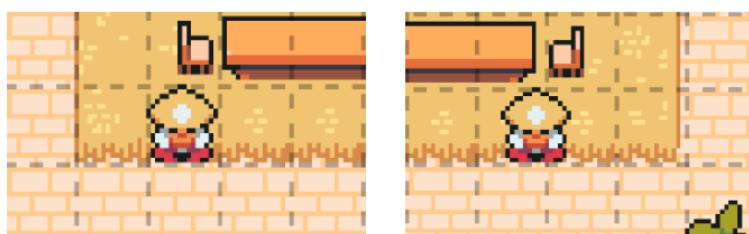
1st Test: Second Iteration Test Results

1st Test: Second Iteration Video.

The player seems to move faster than before and I have fixed the freezing issue, however, I have some other issues, which are:

Issues:

1. When moving to certain tiles, the player sprite becomes blurry.
2. When the player travels, the player moves out of the grid when speed is toggled on.



In grid

Out of grid

Reason:

One of the main reasons for this issue is that `this.speed` needs to be an integer value rather than a decimal number. This is because the `x` and `y` coordinates of each tile are whole numbers, and the player cannot stand in a tile with their `x` and `y` coordinates being decimal numbers. However, as I set `this.speed` to 1.5 (A decimal number), the `this.tilesLeft` and the player coordinates also become a decimal number whenever the player travels, as I have multiplied or subtracted from these attributes using a decimal value. As a result, the `x` and `y` coordinates become decimal numbers, meaning that the player is now out of grid as the player coordinates do not match the coordinates currently on.

Solution: Restrict `this.speed` to integers only, and avoid using float numbers or any other decimal for this attribute.

1st Test: Third Iteration

I have set speed to an integer such as 2. This should fix the issue of blurriness and the grid movement.

```
window.Maps = {
  StartingHouse {
    //...
    entities: { //Stores every entity for this map
      player: new Player({
        x: 5, y: 4,
        name: "RedSamurai",
        speed: 2,
      })
    }
  }
}
```

1st Test: Third Iteration

The test was successful and now I am finally able to configure the speed of the player. We can now expand and build on this feature by allowing the player to toggle between the speed boost and walking at a normal speed.

Speed Toggle

To add a speed toggle feature, I will have to create a new speedBoolean, which sets to true if the player is holding the “ShiftLeft” and sets back to false if the player releases “ShiftLeft”. To implement this, I will need to go back to the **keyInput.js** file and add this attribute to the **keyInput class**:

```
class keyInput{
    constructor(){

        // Initialize an empty array to track held keys
        this.keysHeld = [];
        this.speedBoolean = false;

        // Define a map that associates key codes with directions
        this.keyDirectionMap = {
            "ArrowUp": "up",
            "ArrowDown": "down",
            "ArrowLeft": "left",
            "ArrowRight": "right",

            "KeyW": "up",
            "KeyS": "down",
            "KeyA": "left",
            "KeyD": "right"
        };
    }
}
```

keyInput.js

I have added at **this.speedBoolean** attribute which is also a boolean that determines whether the player has toggled on their speed boost or not.

True: The player has activated the speed boost feature

False: The player has deactivated or is not using the speed boost feature

```

class keyInput{
    //...
    handleKey(event){
        //Gets the direction value from input key
        const direction = this.keyDirectionMap[event.code];
        //Gets the index of the direction in the keysHeld array
        const index = this.keysHeld.indexOf(direction);

        //Checks if the user presses a key
        if(event.type === 'keydown'){
            //If the held key is not in the keysHeld array...
            if(!this.keysHeld.includes(direction) && direction){
                //...Then add the direction to the front of the keysHeld array
                this.keysHeld.unshift(direction);
            }
            //If the held key is a Shift button and the speedBoolean is false...
            if(event.code === 'ShiftLeft' && this.speedBoolean === false){
                //... Then set the speedBoolean as true
                this.speedBoolean = true;
            }
        }

        //Checks if the user releases a key
        else if (event.type === 'keyup'){
            //If the released key is in the keysHeld array...
            if(index > -1 && direction){
                //...Then remove the direction of the corresponding key
                this.keysHeld.splice(index, 1);
            }
            //If the released key is a Shift button & the speedBoolean is true...
            if(event.code === 'ShiftLeft' && this.speedBoolean === true){
                //... Then set the speedBoolean as false
                this.speedBoolean = false;
            }
        }

        //...
        //This function fetches whether the player is running or not
        get speedBoost(){
            return this.speedBoolean;
        }
    }
}

```

keyInput.js

I have made major changes to the keyInput class, specifically the handleKey() method. I have changed the if statement where it checks if the player input event is a keydown event, then it checks whether the input is not already in the **this.keyHeld** list and if is a direction key, if these conditions are met, then the new direction input is set to the front of the array.

The actual changes and the code that contributes to the toggling of the speed boost using shift begins where the new piece of code determines if the user is holding or has released the left Shift key and updates the **speedBoolean** attribute as needed. If the key is pushed when **speedBoolean** is currently false, it is set to true. If the key is released while **speedBoolean** is still set to true, **speedBoolean** is set to false.

This keyInput class's handleKey() function and speedBoost getter provide a clean and efficient way to handle keyboard inputs and determine if the player is holding down the shift key. I could use a different approach by simply adding an event listener to the global scope, however, my approach is better because it provides encapsulation and separation of responsibilities, enabling the remainder of the code to focus on game logic rather than event handling.

```
//Draws every single entity
Object.values(this.map.entities).forEach(entity => {
    entity.update({
        direction: this.directions.direction,
        speedBoost: this.directions.speedBoost
    })
    entity.sprite.drawObj(this.context);
})
```

Game.js

The speedBoost attribute of the directions object is passed to the update method of each entity, in this case is only going to affect the player instance rather than other entities. This property specifies whether or not the player is running, and it allows entities to alter their movement speed accordingly.

Additionally, passing a single object with several characteristics (direction and speedBoost) is more efficient than passing them individually. It reduces the amount of parameters in the code and provides a centralised area for storing and accessing all input-related information.

```

class Player{
    //...
    //Updates the character in each Loop
    update(state) {
        this.updatePos();
        //if speedBoost is true and the player's speed is default
        if (state.speedBoost && this.speed === 1){
            //...Then give the player a speed boost
            this.speed = 2;
        //Otherwise, if speedBoost is false and the player's speed is
        boosted
        } else if (state.speedBoost === false && this.speed === 2){
            //...Then return player speed to default
            this.speed = 1;
        }
    }
}

```

Player.js

The first if statement checks if there are no more tiles left to travel, if that condition is met, then determines whether or not to give the player a speed boost based on the state.speedBoost value, which checks if the player is holding shift Left or not, and the current speed of the player. This method achieves the effect of increasing the player's speed without adding excessive complexity to the code. Setting distinct speed and boosted attributes for the player, or using a separate method to handle the speed boost logic, are alternative approaches that may need more code and be more difficult to grasp.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if pressing ShiftLeft key changes the speed boolean	this.speedBoolean	The console should display “speed boost is on” if ShiftLeft is held, and display “speed boost is off” if ShiftLeft is released.
2	Check if the player’s speed changes when speed boost is toggled on and off	this.speedBoolean	The character should travel faster if ShiftLeft key is held, The player should travel slower if the ShiftLeft key is released.
3	Checking if the player is able to toggle speed boost while travelling	this.speedBoolean this.TilesLeft Player.update()	The character’s speed should change when while travelling if ShiftLeft is held or released

The first test is a **normal** test which determines whether the shift key’s functionality in updating the **this.speedBoolean** attribute. When the player holds the ShiftLeft key, the game should be able to identify that the player wants to toggle this speed boost feature, to indicate this I am going to use the usual console.log() function which will let me know if the player has toggled the speed boost or not.

The second test is a **normal** test which determines whether the speed value of the **this.speedBoolean** attribute changes the rate of how many tiles the player is travelling. This test is the main test that emphasises the functionality of this speed boost feature.

The final test is an **erroneous** test which determines whether the player is able to toggle the speed boost while travelling, meaning that the **this.TilesLeft** will be somewhere between 16 and 1, and I need to verify if the speed boost is free of errors when the speed boost is activated in this situation. I could not test this, however, this is important because not testing this may lead to this feature to be incomplete and worse user experience, as the user may not be able to toggle this speed boost feature at all.

1st Test: First Iteration

```

class keyInput{
    //...
    handleKey(event){
        //Gets the direction value from input key
        const direction = this.keyDirectionMap[event.code];
        //Gets the index of the direction in the keysHeld array
        const index = this.keysHeld.indexOf(direction);
        console.log(event.code, event.type)
        //...
        if(event.code === 'ShiftLeft' && this.speedBoolean === false){
            //... Then set the speedBoolean as true
            this.speedBoolean = true;
            console.log("Speed boost is on!")
        }
        //...
        if(event.code === 'ShiftLeft' && this.speedBoolean === true){
            //... Then set the speedBoolean as false
            this.speedBoolean = false;
            console.log("Speed boost is off")
        }
    }
    //...
}

```

The first console.log function will display what key and the type of event is the player interacting with. This interaction will determine if the player is activating speed boost or not, therefore I added two more console.log functions to check if the event that the player is interacting with matches up with the current speed toggle.

1st Test: First Iteration Test Results

```

ShiftLeft keydown
Speed boost is on!
ShiftLeft keyup
Speed boost is off
ShiftLeft keydown
Speed boost is on!
ShiftLeft keyup
Speed boost is off
ShiftLeft keydown
Speed boost is on!

```

As you can see, when the ShiftLeft key is held, the player then toggles on speed boost. When the ShiftLeft key is released, the speed boost is toggled off.

Therefore, **the test was successful.**

2nd Test: First Iteration

```
class keyInput{
    //...
    handleKey(event){
        //...
        if(event.code === 'ShiftLeft' && this.speedBoolean === false){
            //... Then set the speedBoolean as true
            this.speedBoolean = true;
            console.log("Speed boost is on!")
        }
        //...
        if(event.code === 'ShiftLeft' && this.speedBoolean === true){
            //... Then set the speedBoolean as false
            this.speedBoolean = false;
            console.log("Speed boost is off")
        }
    }
}
```

keyInput.js

Similar to the previous test, I will be required to also log when the user is toggling between the speed boosts, which will help me determine when the speed boost should be activated and detect for any visual changes in the speed of the player.

2nd Test: First Iteration Test Results

[2nd Test: First Iteration Video](#)

The test was successful, as shown in the video, the player's speed changes when speed boost is toggled on and off. Additionally, the player travels faster if ShiftLeft key is held, And travels slower when the shiftLeft key is released, which is indicated by the console

3rd Test: First Iteration

3rd Test: First Iteration

[3rd Test: First Iteration Video](#)

The test was unsuccessful, because the player freezes after the player moves for a certain amount of tiles

Issues:

1. Character Freezes
2. If the player is successfully able to toggle speed boost while travelling, the character becomes out of grid.

Debugging #1

I doubt it has to do with this.TilesLeft, so we need to check this attribute when updates are made on the character

```
updatePos() {
    if (this.TilesLeft > 0) { //If the player has to move
        //Checks which direction it needs to move to
        const [axis, value] = this.directionDict[this.direction]
        //Changes their position value on the correct axis
        this[axis] += value*this.speed
        //The method will stop when the TilesLeft is 0
        this.TilesLeft -= this.speed
        console.log(this.TilesLeft)
    }
}
```

This debug test will allow me to figure out what is the issue with this speed boost feature and why the player seems to freeze.

Debug Test Results

0	Player.js:99
15	Player.js:99
14	Player.js:99
13	Player.js:99
12	Player.js:99
11	Player.js:99
10	Player.js:99
9	Player.js:99
8	Player.js:99
7	Player.js:99
6	Player.js:99
5	Player.js:99
Speed boost is on!	Player.js:65
1	Player.js:99
-3	Player.js:99

As seen in this console snippet, when the player toggles the speed boost feature when the **this.TilesLeft** is not 0, meaning that the player is currently in the midst of travelling, the **this.TilesLeft** will eventually become negative

Reason:

When speed boost is toggled on midway when travelling, **this.TilesLeft** is simultaneously multiplied with **this.speed** which has been boosted to 2.

Solution:

The speed boost will be activated when the character's has no more tiles left to travel, So when **this.TilesLeft = 0**

3rd Test: Second Iteration

```

class Player{
    //...
    //Updates the character in each Loop
    update(state) {
        this.updatePos();

        //If there are no more tiles left to travel...
        if(this.TilesLeft === 0){
            //if speedBoost is true and the player's speed is default
            if (state.speedBoost && this.speed === 1){
                //...Then give the player a speed boost
                this.speed = 2;
            //Otherwise, if speedBoost is false and the player's speed is
            //boosted
            } else if (state.speedBoost === false && this.speed === 2){
                //...Then return player speed to default
                this.speed = 1;
            }
        }
    }
}

```

Player.js

I wrapped this code inside an if statement which will only run the following code to toggle on the speed boost feature, if this.TilesLeft is 0.

3rd Test: Second Iteration Test Results

[3rd Test: Second Iteration Video](#)

The test was successful, the player is able to toggle speedBoost on and off while travelling between tiles as seen in the video. I have also met the following success criteria:

Success Criteria	Evidence	Justification
Speed Boost	Video of the user toggling on speed boost allowing the character to travel at a faster speed	The third criteria ensures that the user is able to toggle speed boost abilities, which allows the user to perform more actions, faster travel, and an increased sense of control for the user over their character's movement. I could not add this speed boost feature to keep my game simplistic and reduce game mechanics complexity. However, adding the speed boost feature will add an additional layer of gameplay depth and allows for more strategic movement options.

Animation Movement

Sprite Direction Configuration

```

class Sprite{
    constructor(config){
        //...
        this.animationSet = config.animationSet || "idle-down";
    }
    //This method takes in a parameter the entity direction
    updateSpriteSet(walkingDir){
        //Then sets to the current this.animationSet attribute
        this.animationSet = walkingDir
    }
    //Sprite Methods
    drawObj(context){
        const {x, y} = this.Obj; //Destructuring
        if (this.isLoaded) { //If the sprite is loaded
            const fx = this.animationsMap[this.animationSet][0][0]*16
            const fy = this.animationsMap[this.animationSet][0][1]*16
            //then draw the sprite on the game canvas
            context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
        }
    }
    //...
}

```

Entity.js

I have added a new method to the sprite class that will allow me to change the sprite animation set when the 'walkingDir' parameter is passed on.

```

//Player movement animation
this.animationsMap = config.animationsMap || {
    "idle-down" : [ [0,0] ],
    "idle-up"   : [ [1,0] ],
    "idle-left" : [ [2,0] ],
    "idle-right": [ [3,0] ],
    "walk-down" : [ [0,1], [0,2], [0,3], [0,0] ],
    "walk-up"   : [ [1,1], [1,2], [1,3], [1,0] ],
    "walk-left" : [ [2,1], [2,2], [2,3], [2,0] ],
    "walk-right": [ [3,1], [3,2], [3,3], [3,0] ]
}

```

The `animationSet` attribute is used to fetch the sprite frames from the entity sprite sheet.

In order to fetch these individual frames for the respective animation, the code needs to identify which animation set it needs to draw.

To sort this issue, we can modify the `drawObj` method in the sprite class, where I have added `fx` and signifying the coordinates of the current frame that is being drawn.

The function will only draw the first sprite in the spritesheet for the corresponding direction of the entity, for now, I will add proper animations later on.

```

class Player{
    update(state){
        updatePos();
        updateSprite();
        //...
    }
    //...

//Updates the player's sprite
updateSprite(){
    //If there are no tiles left to travel...
    if(this.tilesLeft === 0){
        //...Then set the player's sprite animation to 'idle' + direction
        this.sprite.updateSpriteSet("idle-"+this.direction)
    } else {
        //...Or else, set it to 'walk' + direction
        this.sprite.updateSpriteSet("walk-"+this.direction)
    }
}
}

```

Player.js

Furthermore, I have added a new method to the player class; “updateSprite()” which is called in the update() method of the player. The purpose of this function is to update the sprite based on the player’s current behaviour.

If it is standing still, meaning that the player has no tiles left to travel, this function will take their direction and put it in a string format attached at the end of the “idle-” string.

This string is used as a parameter for the sprite.updateSpriteSet() method where the spirit of the player will be set appropriately.

On the other hand, if the player is moving, meaning that it has tiles left to travel, the function will do the same thing, the only difference being that the string that is being passed as a parameter for sprite.UpdateSpriteSet is going to be “walk-” rather than “idle-”, telling the code to switch to the walking animation set. However, I haven’t added walking animations yet. That feature will be implemented after I am able to successfully configure the player sprite based on their current behaviour.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if character sprite changes respect to the direction	this.sprite.currentSpriteSet	The character sprite set should change depending on the direction they're facing. if the character is facing right, the console should print "idle-right"
2	Checking if character sprite set changes to walk sprite set when travelling	this.sprite.currentSpriteSet	The character sprite set should change depending on the direction they're travelling. if the character is travelling right, the console should print "walk-right"
3	Check if the player can still toggle speed boost while changing sprite sets	this.speedBoolean this.sprite.currentSpriteSet	When the player has activated speed boost, there shouldn't be any errors with the player sprite direction animation set

The first test is a **normal** test that determines whether the entity spruce changes respect to their direction. I need to test this by changing the **this.sprite.currentSpriteSet** which is meant to update depending on the direction and the key inputs that I am going to test with. This test is important as it is a fundamental test which will help me to verify If I could continue developing with the animation movement system

The second test is also a **normal** test that determines whether the entity's sprite set is updated from the walking animation when the player starts travelling. I am again using **this.sprite.currentSpriteSet** attribute for the purposes of this test which should update depending on the player's movement. This test is important as it is a stepping stone to allow for further development on other animation systems

The third test is a **normal** test that also implements the previous feature, which is the speed boost feature, to determine whether both features are compatible with each other and to check whether when speed boost is toggled on, the animation speed changes as well.

1st Test: First Iteration

```
class Sprite {
//...
//Sprite Methods
drawObj(context){
    const {x, y} = this.Obj; //Destructuring
    if (this.isLoaded) { //If the sprite is Loaded
        const fx = this.animationsMap[this.animationSet][0][0]*16
        const fy = this.animationsMap[this.animationSet][0][1]*16
        //then draw the sprite on the game canvas
        context.drawImage(this.skin, fx, fy, 0, 16, 16, x, y, 16, 16)
    }
}
}
```

In this test, I am testing if the player direction changes. Since I am not testing for any walking animation, I can set fy to 0 instead to simplify this test.

```
class Player{
//...
//Updates the player's sprite
updateSprite(){
    //If there are no tiles left to travel...
    if(this.tilesLeft === 0){
        //...Then set the player's sprite animation to 'idle' + direction
        this.sprite.updateSpriteSet("idle-"+this.direction)
        console.log(this.sprite.currentSpriteSet)
    } else {
        //...Or else, set it to 'walk' + direction
        this.sprite.updateSpriteSet("walk-"+this.direction)
    }
}
//...
}
```

Player.js

The `console.log()` function will display which direction the player is currently facing while it is travelling which will help me verify if the character sprite corresponds to the respective direction.

1st Test: First Iteration Test Results[1st Test: First Iteration Video](#)

```

57 idle-down
36 idle-right
34 idle-down
30 idle-left
20 idle-up
12 idle-left
13 idle-up
13 idle-right
19 idle-down
2 idle-left
10 idle-down

```

The player direction sprite seems to be working fine, as the direction that is shown on the console matches the current direction of where the player is travelling.

The test was successful.

2nd Test: First Iteration

```

class Player{
//...
//Updates the player's sprite
updateSprite(){
    //If there are no tiles left to travel...
    if(this.tilesLeft === 0){
        //...Then set the player's sprite animation to 'idle' + direction
        this.sprite.updateSpriteSet("idle-"+this.direction)
    } else {
        //...Or else, set it to 'walk' + direction
        this.sprite.updateSpriteSet("walk-"+this.direction)
        console.log(this.sprite.currentSpriteSet)
    }
}
//...
}

```

Player.js

In this test, I am checking if the player animation changes to a walking animation set when travelling and sets back to idle animations when the player is not travelling..

2nd Test: First Iteration Test Results**2nd Test: First Iteration Video****Walking****Standing**

The player seems to be successfully switching between the walking animation to the idle animation depending on their respective behaviour as seen in the video test.

The test was successful.

However, I will still need to add walking animations; when the player travels, there's an animation of the character sprite walking. The purpose of the 2nd test is to clarify if the game can detect which animation sprite to use when the player is walking or is idle.

This test will facilitate the next process of adding frame by frame walking animation.

3rd Test: First Iteration

```
class Player{
    //...
    //Updates the character in each Loop
    update(state) {
        this.updatePos();
        this.updateSprite();

        //If there are no more tiles left to travel...
        if(this.tilesLeft === 0){
            //if speedBoost is true and the player's speed is default
            if (state.speedBoost && this.speed === 1){
                //...Then give the player a speed boost
                this.speed = 2;
                console.log("Speed Boost is on!")
            //Otherwise, if speedBoost is false and the player's speed is boosted
            } else if (state.speedBoost === false && this.speed === 2){
                //...Then return player speed to default
                this.speed = 1;
                console.log("Speed Boost is off")
            }
        }
    }
}
```

```
//Updates the player's sprite
updateSprite(){
    //If there are no tiles left to travel...
    if(this.tilesLeft === 0){
        //...Then set the player's sprite animation to 'idle' + direction
        this.sprite.updateSpriteSet("idle-"+this.direction)
    } else {
        //...Or else, set it to 'walk' + direction
        this.sprite.updateSpriteSet("walk-"+this.direction)
    }
    console.log(this.sprite.currentSpriteSet)
}
//...
}
```

Player.js

In this test, I am checking if the player can still toggle speed boost while changing directions. I have added a few temporary console.log() functions to test this.

The first two console.log() functions in the Player class checks whether the speed boost is toggled on and off. The second console.log() function only checks the current player direction animation set.

3rd Test: First Iteration Test Results

3rd Test: First Iteration Video

```
7 walk-down
7 idle-down
7 walk-down
7 idle-down
speed boost is off.
20 idle-down
15 walk-left
15 idle-left
15 walk-left
15 idle-left
speed boost is on!
7 walk-left
7 idle-left
7 walk-left
```

As seen in the video, the player's sprite changes when directions are switched while speed boost is toggled on. Therefore, **the test was successful**.

Sprite Frame Animation

```
class Sprite{
    //...
    //Initialises attributes for the walking animations...
    this.animationSet = config.animationSet || "idle-down";
    //Initial sprite frame that is currently being drawn
    this.currentSpriteFrame = 0;
    //Maximum number of iterations to draw the currentSpriteFrame
    this.framesLimit = 8;
    //Frames left to draw
    this.framesLeft = this.framesLimit;
}
```

I have added new attributes in addition to the Sprite class which will allow me to add and configure sprite animations. I have added a **currentSpriteFrame** attribute, initially set to zero, this attribute will store the current sprite frame that the entity is on at the moment, and this sprite frame will be drawn for multiple times, which depends on '**framesLimit**', this attribute will store the maximum number of loops to draw the **currentSpriteFrame** will be drawn. So in this case, the **currentSpriteFrame** will be drawn 8 iterations of the game loop. To measure how many iterations has the currentSpriteFrame been drawn for, I have added a new attribute called '**framesLeft**'. Similarly to **TilesLeft**, this attribute will measure how many more iterations are left until the game can draw the next frame of the **animationSet** using the **animationMap** dictionary.

```
//updates the current sprite set if there is a change to their direction
updateSpriteSet(walkingDir){
    //Sets the current animation set to the player's direction
    this.animationSet = walkingDir
    //Resets the number of frames Left to draw
    this.framesLeft = this.framesLimit;
}

updateFramesLeft(){
    //If there are frames Left to draw...
    if(this.framesLeft > 0){
        //Decrement it
        this.framesLeft--;
        //Return back to drawObj()
        return;
    }
}
```

```

    //Otherwise reset the frames Left to draw
    this.framesLeft = this.framesLimit;
    //Go to the next sprite frame that needs to be drawn
    this.currentSpriteFrame += 1;
}

//Sprite Methods
drawObj(context){
    const {x, y} = this.Obj; //Destructuring
    if (this.isLoaded) { //If the sprite is loaded
        const fx =
this.animationsMap[this.animationSet][this.currentSpriteFrame][0]*16
        const fy =
this.animationsMap[this.animationSet][this.currentSpriteFrame][1]*16
        //then draw the sprite on the game canvas
        context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
    }
    this.updateFramesLeft();
}
}

```

Entity.js

I have added major changes to the sprite class in **Entity.js**.

The first change that I have implemented in the updateSpriteSet method, which is the main purpose of this method is to change the animation direction when the entity changes the direction of their movement, now that I need to take into account the movement animation, I will also be required to reset the frame progress attribute for this class, which is the **this.framesLeft** attribute, by setting it back to the **this.framesLimit**. This will reset back the frames left for the game to draw and avoid any errors in the animation movement.

The **updateFramesLeft** which updates the progress of the animation by decrementing the **this.framesLeft** to move on to the next animation frame until it equals zero, and after this process it resets the value of the **this.framesLeft** attribute and moves on to the next frame by incrementing the **this.currentSpriteFrame**.

I have changed the fx and fy to handle further frames in the sprite sheet rather than only the first sprite in the spritesheet which will allow the animation features to execute itself.

I could use a sprite animation library to handle the sprite animations in my game instead of using all these functions and complex logic. The library will allow me to handle animation frames and timing much easier. However, using a library may not provide the customisation that is going to be needed for all the different types of animations, characters and sprite sheets. Furthermore, my approach is straightforward and maintainable to handle sprite animations for this type of game.

Adding speed boost to walking animation

```

updateFramesLeft(){

    //If speed boost is on...
    if(this.Obj.speed === 2){
        //...then set the frames limit to 4 (frames will generate faster)
        this.framesLimit = 4;
    } else {
        //...or else set the frames limit to default
        this.framesLimit = 8;
    }

    if(this.framesLeft > 0){
        this.framesLeft--;
        return;
    }

    if(this.animationsMap[this.animationSet][this.currentSpriteFrame] ===
undefined){
        this.currentSpriteFrame = 0;
    }
}

```

I have made an additional change to the new UpdateFramesLeft() method.

When the player toggles speed boost, the frameLimit should be set to '4', meaning that each frame is going to be iterated only for 4 game loops rather than 8 when speed boost is toggled on, therefore, this would give the illusion that the player is walking faster as each frame in the animation is drawn for less amount of time.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the frame animation works one of the animation set	this.animationSet	The character's sprite should be iterate through "walk-down" correctly without skipping frames, freezing or any other errors
2	Check if speed boost is turned on, the animation should become faster	this.speedBoolean	The character sprite animation should iterate through the animation much faster when speed boost is on
3	Check if the walking animation changes when the character travels in a different direction	this.updateSpriteSet	The character walking animation should correspond to the current direction that the entity is travelling.

1st Test: First Iteration

```
class Player{
    ...
    //Updates the character in each Loop
    update(state) {
        this.updatePos();
        this.updateSprite();
    }
    ...
}
```

In the player class, I have commented the updateSprite() method off, for the purposes of this test. In this test, I am only checking if the player "walk-down" animation works perfectly.

```
class Sprite{
    ...
    //Initialises attributes for the walking animations...
    this.animationSet = "walk-down";
    //Initial sprite frame that is currently being drawn
    this.currentSpriteFrame = 0;
    //Maximum number of iterations to draw the currentSpriteFrame
    this.framesLimit = 16;
    //Frames left to draw
    this.framesLeft = this.framesLimit;
}
```

In sprite class, I have added “walk-down” for the sprite set for testing my code. I am expecting the character to do a “walk down” animation while staying in one position, this test will allow me to check for any errors when the game is iterating through an animation set such as “walk-down”.

1st Test: First Iteration Test Results

[1st Test: First Iteration Video](#)

There seems to be an error where the character iterates through one animation set then the game freezes. A frame becomes undefined at a certain point.

```
* ▶ Uncaught TypeError: Cannot read properties of undefined      Entity.js:75
  (reading '0')
    at Sprite.drawObj (Entity.js:75:81)
    at Game.js:30:25
    at Array.forEach (<anonymous>)
    at Game.js:25:42
```

The test was unsuccessful. - However, this issue will be easily solvable if I debug and test the new additions.

Debug #1

```
class Sprite{
  //...
  //Sprite Methods
  drawObj(context){
    const {x, y} = this.Obj; //Destructuring
    if (this.isLoaded) { //If the sprite is Loaded
      const fx =
        this.animationsMap[this.animationSet][this.currentSpriteFrame][0]*16
      const fy =
        this.animationsMap[this.animationSet][this.currentSpriteFrame][1]*16
      //then draw the sprite on the game canvas
      context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
    }
    this.updateFramesLeft();
    console.log('['+fx/16+', '+fy/16+']');
  }
}
//...
```

To check this issue, I need to check the coordinates of the sprite that is drawing respect to the spritesheet. Therefore, I need to `console.log()` the frame coordinates for the **animationSet** iterations.

Results:

```

17 [0,1]                                         Entity.js:82
17 [0,0]                                         Entity.js:82
17 [0,3]                                         Entity.js:82
17 [0,0]                                         Entity.js:82

✖ ► Uncaught TypeError: Cannot read properties of Entity.js:75
  undefined (reading '0')
    at Sprite.drawObj (Entity.js:75:81)
    at Game.js:30:25
    at Array.forEach (<anonymous>)
    at Game.js:25:42

```

After analysing the following error message and looking through my code, I have figured out the solution for this issue.

Issue:

```

10      //Player movement animation
11      this.animations = config.animations || {
12          "idle-down" : [ [0,0] ],
13          "idle-up"   : [ [1,0] ],
14          "idle-left" : [ [2,0] ],
15          "idle-right": [ [3,0] ],
16          "walk-down" : [ [0,1], [0,0], [0,3], [0,0] ],
17          "walk-up"   : [ [1,1], [1,0], [1,3], [1,0] ],
18          "walk-left" : [ [2,1], [2,0], [2,3], [2,0] ],
19          "walk-right": [ [3,1], [3,0], [3,3], [3,0] ]
20      }

```

The console gives us the correct results, the code iterates through each sprite frame in the ‘walk-down’ sprite set, there are 4 sprite frame for this animation, however, when the code looks for the 5th sprite, it won’t find it so it sets the frame as undefined.

Solution:

I need to handle the fx or fy coordinates when it gets out of the spritesheet range.

If the frame coordinates gets out of the range, I will need to set **currentSpriteFrame** back to 0 (initial sprite in the **animationSet**)

1st Test: Second Iteration

```
updateFramesLeft(){
    //If there are frames left to draw...
    if(this.framesLeft > 0){
        //Decrement it
        this.framesLeft--;
        //Return back to drawObj()
        return;
    }
    //Otherwise reset the frames left to draw
    this.framesLeft = this.framesLimit;
    //Go to the next sprite frame that needs to be drawn
    this.currentSpriteFrame += 1;
    if(this.animationsMap[this.animationSet][this.currentSpriteFrame]
    === undefined){
        this.currentSpriteFrame = 0;
    }
}
```

1st Test: Second Iteration Test Results

[1st Test: Second Iteration Video](#)

The test was successful after the second iteration, the player completed the full “walk-down” animation correctly. Therefore, we can assume that it can iterate through all the other animations as well.

2nd Test: First Iteration

```
update(state) {//Updates the character in each Loop
    //If there are no more tiles left to travel...
    if (this.tilesLeft === 0) {
        //if speedBoost is true and the player's speed is default
        if (state.speedBoost && this.speed === 1) {
            this.speed = 2;
            //...Then give the player a speed boost
            console.log("Speed boost is on!")
            //Otherwise, if speedBoost is false and the player's speed is boosted
        } else if (state.speedBoost === false && this.speed === 2) {
            //...Then return player speed to default
            this.speed = 1;
            console.log("Speed boost is off!")
        }
    }
}
```

Player.js

As we saw a similar logic before, I will be using the console logs functions again to show that the speed boost is toggled on at random times, and when it is toggled on, I need to verify if the speed of the animation is changing accordingly.

2nd Test: First Iteration Test Results

[2nd Test: First Iteration Video](#)

The test was successful as seen in the video, when the player toggles on the speed boost, more frames are iterated and drawn faster giving the illusion that the entity is moving faster.

3rd Test: First Iteration

In this test, I am checking if the walking animation changes when the character travels in a different direction. When the player travels a tile in a different direction, the game must ensure that the player executes a walking animation correctly for the corresponding direction. If these conditions are met, then the test can be declared successful. I will be required to use the **this.updateSprite()** method again for the purposes of this test.

3rd Test: First Iteration Test Results

[3rd Test: First Iteration Video](#)

The test was unsuccessful.

Issue:	The character does not iterate through all 4 frames of animation for any direction. When the player moves, the sprite for the walking direction is not properly iterated through, resulting in the character sprite having errors with animation and only iterating the first two frames.
Reason:	The animation frames are being constantly reset every time the player travels a tile. Meaning that every time the entity moves, the sprite set animation frame progress for the walking direction is set back to the first frame, forcing the animation to restart and preventing from completing the full animation.
Solution:	In the updateSpriteSet() method, It should only reset the animation if the new player's walking direction is different from the current walking direction (which we need to implement in the second iteration of this test) or when it has finished drawing all the 4 frames (which is already implemented and tested)

3rd Test: Second Iteration

```
updateSpriteSet(walkingDir){
    if(this.animationSet !== walkingDir){
        this.framesLeft = this.framesLimit;
        this.currentSpriteFrame = 0;
        this.animationSet = walkingDir
    }
}
```

I have changed this method to first check if the parameter “walkingDir” is already set to animationSet. If not, then it means the player has switched their direction or their behaviour (walk or idle), therefore, reset the frame animation to start by setting framesLeft to the current limit and set currentSpriteFrame back to 0.

After all of that process, set the animationSet to the parameter, and now the game is ready to execute the new animation. This should fix the previous issue.

3rd Test: Second Iteration Test Results

[3rd Test: Second Iteration Video Test](#)

The player walking animation seems to be working correctly, I also tested it by toggling speed boost on and off and I can ensure that the animation feature is free from errors.

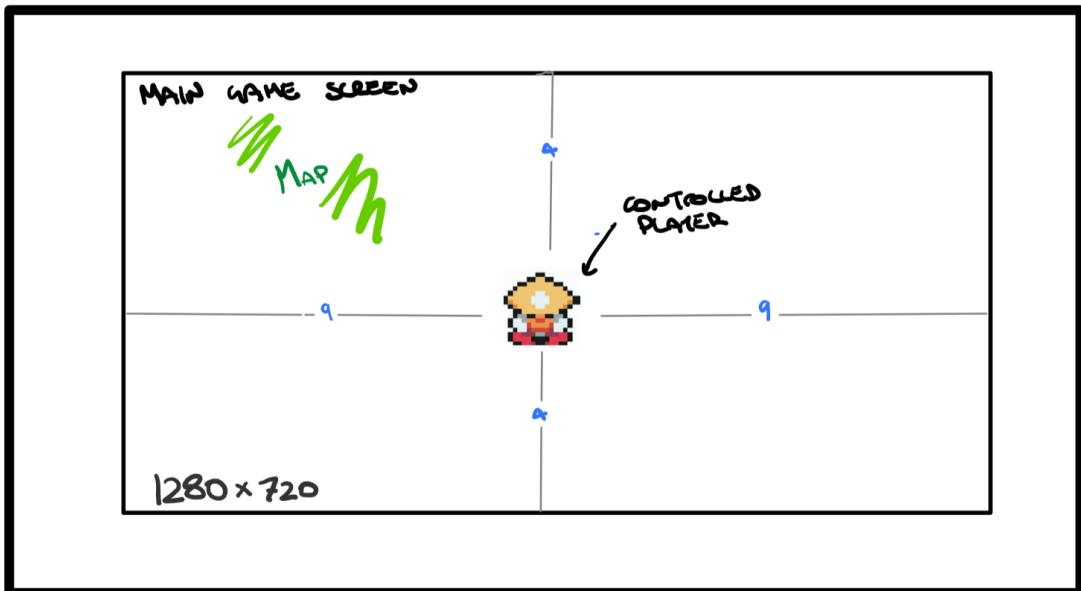
The test was successful and I have also met the following success criteria:

Success Criteria	Evidence	Justification
Character Walking Animation	Video of the character running through a walking animation with the correct frames.	The fourth criteria emphasises on character animation movement which aims to provide players a sense of realism and immersion. I could not add the animations which will allow me to focus on other features instead, as often the animation systems require a long time to develop because of their complexity. However, adding the animation for the character movement because it will add more visual appeal and enhance the user experience, whereas not adding it will make the movement system and the overall game feel dull.

Character Camera

The final sub-feature of the player feature of my game is the character camera.

The camera will follow the player as it travels around the map, it will ensure that the player stays in the centre of the canvas at all times.



To get our player on the centre of the canvas, we need to set their:

The x coordinate: 9 tiles away from the canvas border, right and left.

The y coordinate: 4 tiles away from the canvas border, top and bottom.

To achieve this I can create a camera variable and implement it in the following code

```
//Game Loop
Loop() { //Loop method for the game
  const gameLoop = () => { //Declares a game Loop function
    //Calls a function before going repainting the next frame
    requestAnimationFrame(() => {

      const camera = this.map.entities.player; //Get the camera object
      const entities = Object.values(this.map.entities);

      //Drawing Layers
      this.map.drawLower(this.context, camera);
      this.map.drawCollision(this.context, camera);
    });
  }
}
```

Game.js

To create the camera variable, I have just taken the player and stored it inside this new variable. Then I pass in the camera variable to all the drawing functions.

```
//...
//Draws every single entity
Entities.forEach(entity => {
    entity.update({
        direction: this.keyInput.direction,
        speedBoost: this.keyInput.speedBoost,
    })
    entity.sprite.drawObj(this.context, camera);
})

this.map.drawUpper(this.context, camera);

gameLoop(); //Reiterates the function
})
//...
```

I have also added the camera variable to the entity drawing function, this approach is really easy to implement as the drawing functions and the camera variables are in the same scope which is the game loop.

Adding camera to entities

```
//Sprite Methods
drawObj(context, camera){
    const x = this.Obj.x + 9*16 - camera.x
    const y = this.Obj.y + 4*16 - camera.y
}
```

Entity.js

Now that the variable has been passed, it's time to utilise them in the drawing methods. When drawing the coordinates of the entities, we need to make sure that their drawing coordinates are relative to the player's position hence the calculations when it comes to drawing their coordinates position.

Adding camera to map

```
class Map {
//...
drawLower(context, camera){
    context.drawImage(lowerLayer, 9*16 - camera.x, 4*16 - camera.y);}
drawCollision(context, camera){
    context.drawImage(collisionLayer, 9*16 - camera.x, 4*16 - camera.y);}
drawUpper(context, camera){
    context.drawImage(upperLayer, 9*16 - camera.x, 4*16 - camera.y);}
}
```

Map.js

I have also implemented the same calculations when drawing layers in the map class, so it gives an illusion that the player is centred and the game moves in relative to the player position. However, the surroundings such as entities and map layers are the actual objects that are moving, the player always stays in the same position.

Test

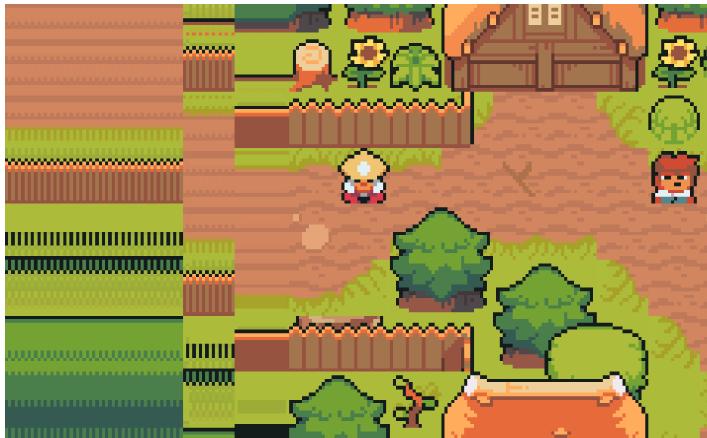
Test No.	What I am testing	Test code	Result Required
1	Check if the camera follows the character	camera Game.Loop()	The camera should provide an illusion of following the character, with no errors.

1st Test: First Iteration

1st Test: First Iteration Test results

[1st Test: First Iteration Video](#)

The test was unsuccessful as I found a few errors, which are mostly visuals.



First Issue

The first issue is that it redraws the images on top of each other over and over again, and it doesn't get cleared every loop.

First Solution

Clear the canvas in every loop to avoid the canvas to repaint again and again on unnecessary sections of the canvas.

Second issue

The upper layer detaches from the rest of the layers when the character is travelling downwards.

This is because we are drawing and updating layers and characters in the same loop.



Second Solution

Separating them into two different loops will solve the problem to ensure we are not drawing and updating the state all at once.

1st Test: Second Iteration

Solution for 1st issue: Clearing the canvas

```
//Game Loop
Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game loop function
        //Calls a function before going repainting the next frame
        requestAnimationFrame(() => {

            //Clears the canvas from previous drawing state
            this.context.clearRect(0,0, this.canvas.width, this.canvas.height)
            const camera = this.map.entities.player; //Get the camera object
            const entities = Object.values(this.map.entities);
            //...
        })
    }
}
```

This space efficient one-liner code should delete everything in the last repaint.

Solution 2: Separating the draw and update loop

```
Loop(){
    //...
    //Draws every single entity
    entities.forEach(entity => {
        entity.update({
            direction: this.keyInput.direction,
            speedBoost: this.keyInput.speedBoost,
        })
    })
    //Drawing Layers
    this.map.drawLower(this.context, camera);
    this.map.drawCollision(this.context, camera);

    entities.forEach(entity => {
        entity.sprite.drawObj(this.context, camera);
    })
    //...
}
```

This loop will draw each entity from the **map.entities** and takes in the camera variable as a parameter.

1st Test: Second Iteration Test Results

[1st Test: Second Iteration Video](#)

The issues have been fixed and the player camera seems to be working properly.

The test was successful, however,

I have also met the following success criteria as I have provided the correct evidence:

Success Criteria	Evidence	Justification
Character Camera	Video of the game where the map moves following the character's movement	An essential feature that must be developed in order to provide players with a clear view of their characters and the map. This feature should follow the character around, always keeping the character in the centre of the canvas. I could instead add a fixed camera system to simplify the game mechanics. However, the dynamic camera is far better because it adds to the immersive nature of the game.

Review and Refactoring #1:

Since the camera variable is using the player instance in the game loop, I could just make a global object for the player, not only this will allow me to reduce the unnecessary complexity in the code, I won't need to pass in the camera variable in every drawing method.

```
window.player = new Player({
  name: "MaskedNinja",
  x: 4, y: 4
})
windowMaps{
  StartingHouse: {
    name: 'StartingHouse',
    entities: { //Collection of entities of StartingHouse map
      player: new Player({ //creates new Player instance
        name: "MaskedNinja",
        x: 5, y: 4, //sets player properties
      }), //set player source
    }
  }
  //...
}
```

Globals.js

I have made the player as a global object so it can be accessed, updated and referenced anywhere in the code and in a global scope. Therefore, I needed to delete the player instance initialised in windowMaps. This new approach will let me add more features to the game such as the player switching maps etc.

```
class Obj {
  constructor(config) {
    this.isPlaying = false;
    //...
  }
  //...
}
```

Entity.js

```
class Person extends Obj{
  constructor(config) {
    super(config);
    this.isPlaying = true;
    //...
  }
  //...
}
```

Player.js

I needed to add a new attribute which will indicate whether an entity is a player or not, this will allow the game to know which entity is the player going to control.

```
//Game Loop
Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game Loop function
        //Calls a function before going repainting the next frame
        requestAnimationFrame(() => {
            //...
            const entities = Object.values(this.map.entities);
            const camera = player;
            entities.push(player);
            // Iterate through each entity in the list
            entities.forEach(entity => {
                const state = {};
                // add the player's direction and speedBoost to the state
                if (entity.isPlayer) {
                    state.direction = this.keyInput.direction;
                    state.speedBoost = this.keyInput.speedBoost;
                }
                // passing in the state for every other entity
                entity.update(state);
            });
        });
    };
}
```

Game.js

Since I have removed the player instance from window.map, the player instance is not part of **this.map.entities** anymore, therefore I need to push the global player instance into the entities list so the game loop can draw and update all the entities including the player. Additionally, I have in fact refactored the entity update logic, following a more maintainable approach by only passing in the '**state**' object, which contains directionInput and SpeedBoost, both of these attributes only affect the player instance. Therefore, if the entity is the player, then their state will include **keyInput.direction** and **keyInput.speedBoost**, otherwise, the state is going to be unused and empty.

I could have pursued the old approach and kept my code inefficient, however, the older approach is arguably better in some specific aspects because that approach did not require the use of more global objects, meaning that there were less dependencies. However, the new refactored approach outweighs the benefits of the older approach because the player instance is the most important object in the game, and it should be accessible from anywhere from the code. This approach will be useful in the future when I will need to add collisions even for the non-player entity, so the state needs to store an attribute called **state.map**, which is applicable to both the player and other entities such as NPCs and monsters.

Character Transformation

As mentioned in the design section, I am going to add a special ability, more specifically a transformation ability which will allow the user to perform more actions and make the game feel more complete and complex

```
class Sprite {
  constructor(config)
    this.Obj = config.Obj; //Obj Class
    this.skin = new Image(); //Creates a new image attribute for the sprite
    this.name = this.Obj.name
    this.skin.src = "Characters/" + this.Obj.name + "/SpriteSheet.png"
    if(this.Obj.type === "monster"){
      this.skin.src = "Monsters/" + this.Obj.name + "/SpriteSheet.png"
    }
    this.skin.onload = () => {//When the skin is Loaded
      this.isLoaded = true; //Mark as Loaded
    }
    //...
  }
}
```

Entity.js

I have made some changes to the sprite class. Instead of passing type and name as parameters, I have set these two information as attributes of Obj. This will allow the game to update the sprite when there are changes to the character sprite in the Obj class.

```
Class Obj {
  constructor(config){
    this.name = config.name
    this.type = config.type
    //Creates an attribute for the sprite or skin of the game object
    this.sprite = new Sprite({ Obj: this, animationSet: config.animationSet});
    //...
  }
  //...
}
```

Entity.js

Here's the changes I made to the Obj class according to the changes made in the Entity class. I have added a name and type attribute to the Obj class, and an additional **this.sprite** attribute which now takes a two parameters, the current obj instance and the animationSet which is configured from the global objects in Globals.js

```

class Player extends Obj {
  constructor(config) {
    // ...
    this.type = config.type || "character"
    this.behaviour = "standing";
    this.isPlayinger = true;

    this.originalSprite = {
      name: this.name,
      type: this.type
    }
    this.transform = config.transform
  }

  update(state){
    //...
    if (state.speedBoost && this.speed === 1) {
      //...Then give the player a speed boost
      this.speed = 2;
      this.sprite= this.transform
      console.log("Speed boost is on!")
      //Otherwise...
    } else if (state.speedBoost === false && this.speed === 2) {
      //...Then return player speed to default
      this.speed = 1;
      this.sprite= this.originalSprite
      console.log("Speed boost is off!")
    }
  }
  //...
}
}

```

Player.js

I have added many changes, especially regarding the attributes of the player class. The first attribute is the **this.type** attribute which is a binary between “**character**” and “**monster**”. The **this.behaviour** attribute is also a binary attribute between “**standing**” and “**walking**”. The **this.isPlayinger** attribute which specifies whether the instance is a player or any other entity, since this is a player class, this attribute will be true by default. The **this.originalSprite** and **this.transform** objects store the transformation attributes needed which will allow the player to switch back and forth between the monster and character sprites.

As I mentioned before, in order to toggle on the transformation ability, the player will need to use the same button that toggles on the speed boost ability, meaning that I am combining the transformation and speed boost ability into one feature. I could have made two different mechanics for these two different abilities which will mean that the player will have more actions to perform which will increase their enjoyability of the game experience , however, I do not want to create new mechanics to keep the game simpler and more accessible to a larger audience

Test

Test No.	What I am testing	Test code	Result Required
1	Check the player can become a “monster entity” with the correct animation.	this.transform	The player will have a “monster” sprite entity, with the correct animation.
2	Check if the player can switch from character sprite to a monster sprite.	this.update()	The player should be able to switch between character entity and monster entity by toggling on the previous speed boost by holding a ShiftLeft Key
3	Check if the player can switch from character sprite to a monster sprite while walking.	this.update()	The player should be able to toggle this new feature in the midst of travelling from one tile to another

The first test is a **normal** test that determines whether the player can transform into the monster entity. I am going to test this by configuring the **this.transform** attribute to allow the player to transform into the respective monster entity, according to the **transform.name**. This test is important because it will ensure that the game is capable of handling monster entities and allow the user to control this monster entity.

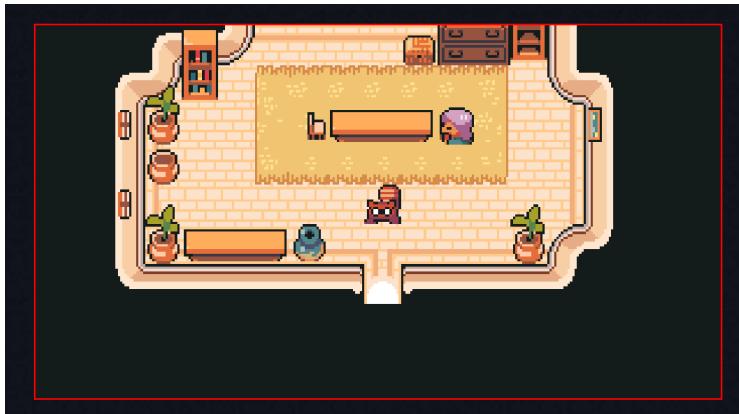
The second test is also a **normal** test that determines whether the player can switch back and forth between their original entity and the monster entity. I won't necessarily require any additional temporary changes for the purpose of this test as everything is already coded, ready to be used and tested.

The third is also a **normal** test that determines whether the game is capable of handling this complex feature of the transformation ability when the player is travelling and toggling the transformation ability simultaneously. This test is important to ensure that this feature is robust and ready to be used in the actual final product of this game.

1st Test: First Iteration

To check if the player can become a “monster entity” with the correct animation, we need to configure the attributes in the player instance in the init() method of game class in Game.js

```
//The init method will Initiate the game
init() {
    //Initiates Player as a global object
    window.player = new Player({
        transform: {
            name: "GoldRacoon",
            type: "monster"
        },
        x: 4, y: 4
    });
}
```



[1st Test: First Iteration Video Test](#)

The test was successful, the player is able to handle a ‘**monster**’ entity animation sprite sheet with no errors

2nd Test: First Iteration

2nd Test: First Iteration Test Results

[2nd Test: First Iteration Video Test](#)

Issue: the **this.Obj.type** and **this.skin.src** is not being updated in the sprite class everytime the player toggles speed boost, as a result, the player’s sprite and skin remains the same.

Solution: The solution is to update the attributes in the sprite class if there is a change in the player’s behaviour such as toggling on speed boost. To implement this we can code a setter and getter methods in the sprite class

2nd Test: Second Iteration

```
class Sprite {
    //...
    set obj(obj) {
        this._obj = obj;
        this.skin.src = "Characters/" + this._obj.name + "/SpriteSheet.png";
        if(this._obj.type === "monster"){
            this.skin.src = "Monsters/" + this._obj.name + "/SpriteSheet.png";
        }
    }

    get obj() {
        return this._obj;
    }
    //...
}
```

The obj property of the Sprite class is set by the set obj() method. It accepts an object as a parameter, sets the _obj property to that parameter, and then sets the src property of the skin image element to the proper file path depending on the name and type of the _obj. If the type is "monster," it will set the src property to the path of the appropriate file in the "Monsters" directory.

Then I added a getter for the _obj property, the get obj() method just returns the value of _obj.

```
class Player extends Obj {
    update(state){
        //...
        if (state.speedBoost && this.speed === 1) {
            //...Then give the player a speed boost
            this.speed = 2;
            This.sprite.obj = this.transform
            console.log("Speed boost is on!")
            //Otherwise...
        } else if (state.speedBoost === false && this.speed === 2) {
            //...Then return player speed to default
            this.speed = 1;
            This.sprite.obj = this.originalSprite
            console.log("Speed boost is off!")
        }
    }
    //...
```

In the update method of the player class, I have changed two lines of code. I have called the obj property of the sprite class to either get or set depending on the **state.SpeedBoost** value.

2nd Test: Second Iteration Test Results

[2nd Test: Second Iteration Test Video](#)

The test was successful as the character can transform into a monster correctly.

3rd Test: First Iteration

Check if the player can switch from character sprite to a monster sprite while travelling, this test is visual and I cannot use screenshots or console log messages to show that I have successfully met the requirements for the functionality of this feature.

3rd Test: First Iteration Test Results

[3rd Test: First Iteration Test Video](#)

The test was successful. The player animation transitions perfectly from character sprite to monster sprite.

I have also successfully met the following success criteria thanks to the evidence provided

Success Criteria	Evidence	Justification
Special Ability	A video of the character using their special ability.	The sixth criteria ensures that the game must have a special ability feature to add an additional layer of gameplay depth and increase the complexity as it allows the character to perform more actions. I could not have a special ability feature to simplify game mechanics and increase game development time to focus on different features. However, a special ability feature allows for more strategic options to the gameplay.

HUD

The HUD feature (Heads-up display) is a key component of the game which will display important information about the current state of the game. I am going to be implementing a HUD for when the player uses their special ability, by transforming into a ‘monster’ entity

They are usually an essential part of most modern video games which allows the player to access information faster and easily without having to pause the game. Additionally, it helps to create a more smooth gameplay experience with a constant stream of the HUD updating.

For example, I can add a specific HUD for the character transformation feature, when the player toggles their special ability and transforms into a monster, a HUD is going to light up showing that the player is currently in ‘monster’ form and their special ability is toggled on. This visual cue should include a sprite frame or an image of the ‘monster’ sprite that the player is currently using. Once the player toggles the special ability off, the HUD should darken back to default. To do this, I can configure the opacity of the HUD showing whether the special ability is toggled on or off.

I could use audio cues to show whether the special ability is toggled on or off. This approach could be useful for users with visual impairments, moreover, audio cues are generally less distracting and intrusive than having a whole image pop up in the game canvas, which could negatively affect the user experience of my game.

However, I am willing to stick to my approach because you can also make the argument for audio cues that may not be suitable for users with hearing impairments. Additionally, most users are more likely to prefer visual indications as it can help to reinforce the game’s theme and aesthetic. Finally, adding audio cues for only one feature will make my game feel inconsistent, if I add a visual cue for character transformation, then I will need to add more audio effects for other features which will cause me to add extra lines of code and it will lengthen the process of this development further. As a result, my original approach of using HUDs is better than adding audio cues.

Firstly, I need to create a new json file named **HUD.js** which will contain the '**HUD**' class

```

class HUD {
    constructor(config) {
        //The name will help to fetch the source
        this.name = config.name
        //The type of HUD of this instance
        this.type = config.type;
        // set default opacity to 1 if not provided
        this.opacity = config.opacity || 1;

    }

    // This function draws the Heads Up Display (HUD) for the game
    drawHUD(context) {
        // Check if the type of the object is a "transform" type
        if (this.type === "transform") {
            // Create a new Image object to hold the HUD image
            this.transformHUD = new Image();
            // Set the source path for the HUD image
            this.transformHUD.src = "HUD/NinePathRect/DialogueBubble2.png"
            // Create a new Image object
            this.image = new Image();
            // Set the source path for the HUD based on its name
            this.image.src = "Monsters/" + this.name + "/SpriteSheet.png"
            // Set the opacity canvas context to the given opacity
            context.globalAlpha = this.opacity;
            // Draw the HUD image on the canvas context at the specified coordinates
            context.drawImage(this.transformHUD, 8, 8)
            // Draw the HUD on the canvas context at the specified coordinates
            // and with the specified width and height
            context.drawImage(this.image, 0, 0, 16, 16, 16, 16, 16, 16)
            // Reset the opacity for the canvas context back to 1
            context.globalAlpha = 1;
        }
    }
}

```

HUD.js

This new fairly small class is a blueprint for every HUD that I am going to include in this game. This class takes in a parameter as a configuration object like most classes, where it stores the configurable attributes such as **this.name**, which is an important string attribute as it will aid to fetch the file path image for the HUD, the **this.type**, will define what type of HUD is the current instance, and finally the **this.opacity** attribute, which is a integer value that indicate how opaque the HUD instance is.

There will only be one HUD method, as this class is really simple, hence why I did not require much planning. The **drawHUD()** is similar to the other drawing methods that I have developed previously, however, with a few changes. This method uses a conditional statement to check if the HUD instance of a 'transform' type, if so, a new image instance is created where the game fetches the filepath of the HUD image using the **this.name** attribute. The game also changes the opacity of the canvas context to the specified value.

The opacity feature of this class is important for the HUD to light on and off to indicate whether the transformation ability has been toggled on and off.

When opacity is 1, the HUD is fully lit up

When opacity is less than 1, the HUD is not lit up but it is still visible

```
const playerHUD = {
    transformHUD: new HUD({
        type: "transform",
        opacity: 0.5
    }),
}

window.player = new Player({
    name: "MaskedNinja",
    transform: {
        name: "GoldRacoon",
        type: "monster"
    },
    x: 4, y: 4,
    hud: playerHUD,
})
```

Global.js

Following a similar pattern to create new instances. I have created a dictionary of HUDs that the player is going to use. This object variable currently only contains '**transformHUD**'.

This approach will allow me to add more HUD easily. The **playerHUD** object is then passed into the **player** instance where the object is passed as an attribute.

```

class Person extends Obj{ //GameObj that can be controlled by the user
constructor(config) {
    super(config); //Inherits methods and attributes from Obj
    //...
    this.hud = config.hud
}

//Updates the character in each Loop
update(state) {
    this.updateSprite(state)
    this.updatePos(state);
    //If there are no more tiles left to travel...
    if (this.TilesLeft === 0) {
        //if speedBoost is true and the player's speed is default
        if (state.speedBoost && this.speed === 1) {
            //...Then give the player a speed boost
            this.speed = 2;
            this.sprite.obj= this.transform
            this.hud.transformHUD.opacity = 0.8
            //Otherwise, if speedBoost is false and the player's speed is boosted
        } else if (state.speedBoost === false && this.speed === 2) {
            //...Then return player speed to default
            this.speed = 1;
            this.sprite.obj= this.originalSprite
            this.hud.transformHUD.opacity = 0.3
        }
    }
}
} //...

```

Player.js

I have added the **this.HUD** attribute which is a object that will store a set of instances from the HUD class, even though I am only going to use one HUD, which is the transformation HUD, indicating to the user whether the transformation ability is toggled on or off, the game is still capable of handling multiple HUDs for one player, one of the benefits of using OOP.

I could have used procedural programming again specifically for the HUD feature and only allow the game to handle one HUD because I am adding only one HUD to my game, however, the OOP approach of creating a class for the HUD feature is more sustainable because it will allow this HUD to be reused later for future development of more complex systems such as, hypothetically speaking, to display how much health a player has left using a classic 'hearts' HUD. Therefore, my approach of handling multiple HUDs is far better.

Moreover, in the player's method, **update()**, I have configured the opacity appropriately to create a lighting up effect for the transform HUD, when the player toggles the transformation ability on and off

```

Class Game{
    //...
    Loop() {
        //...
        //Draw Upper Layer
        this.map.drawUpper(this.context, player
        //Draw player HUD
        Object.values(player.hud).forEach(hud => {
            hud.drawHUD(this.context)
        })
        //...
    }
    //...
}

```

Game.js

I need to make sure to draw all the HUDs of the player. As seen in the code snippet above, I have added this logic, which draws each HUD that the player may have, after the upper layer has been drawn in the game loop, because the HUD should always be on top of everything.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the HUD works	drawHUD()	The HUD should be loaded correctly in the correct position, image and size onto the game canvas.
2	Check if it feature different monsters	Player.transform	The HUD should display other monsters
3	Check if the HUD lights up when the player switches characters	Player.update()	When the player transforms, the HUD should light up by increasing its opacity from 0.3 to 0.8.

This test plan consists of really simple tests, the first test is a **normal** test to check whether the HUD is displayed correctly or not.

The second test is another **normal** test to check whether the HUD class is capable of handling monsters with different types of sprite sheet, with the main purpose of testing its robustness. Finally, the third test is a normal test to see the effect on the HUD when the player switches back and forth using the transformation ability

1st Test: First Iteration

In this test, I need to verify if the HUD is loaded correctly in the correct position, image and size onto the game canvas.



1st Test: First Iteration Test Results

The HUD is being displayed with the correct position, size and image. Therefore the **test is successful**

2nd Test: First Iteration

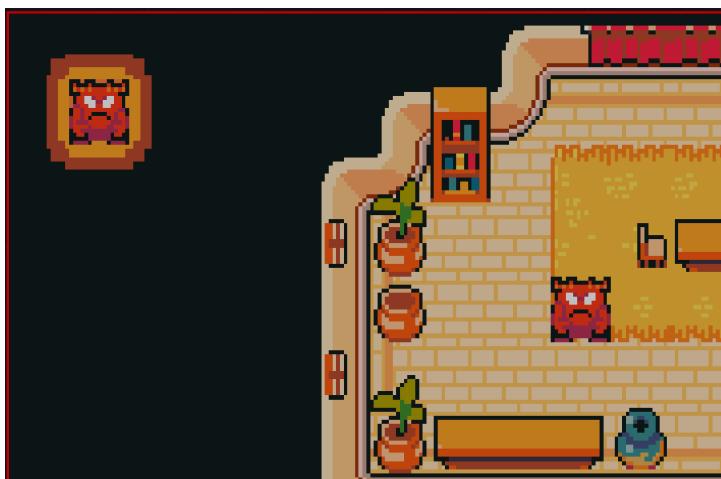
In the second test, I need to ensure that the HUD can handle different monsters as well which may have a different size to the “Racoon” monster.

```
window.player = new Player({
  name: "GreenNinja",
  transform: {
    name: "Beast",
    type: "monster"
  },
  x: 4, y: 4,
  hud: playerHUD,
});
```

I have changed the transform monster into “Beast” rather than “GoldRacoon” to test if it can handle other monsters.

This will allow the user to customise the monster they want to transform into.

2nd Test: First Iteration Test Results



The test seems to be working correctly as the HUD changes to the new ‘monster’ sprite.

The test was successful

3rd Test: First Iteration

In the last test, I need to check if the HUD lights up when the player switches characters.

```
//Updates the character in each loop
update(state) {
    //...
    if (this.tilesLeft === 0) {
        //if speedBoost is true and the player's speed is default
        if (state.speedBoost && this.speed === 1) {
            //...
            this.hud.transformHUD.opacity = 0.8
            console.log("Speed boost is on!")
            //Otherwise...
        } else if (state.speedBoost === false && this.speed === 2) {
            //...
            this.hud.transformHUD.opacity = 0.3
            console.log("Speed boost is off!")
        }
    }
}
```

Player.js

As mentioned previously, I need to test if the opacity changes when the player toggles on the transformation ability, and again, I am using the same console log functions which will output whether the speed boost is on or off to check if the function is actually executed.

If this test fails, meaning that the HUD opacity is not updating, then I can determine whether it's the **update()**'s fault or the HUD class's fault, which will speed up the development process and the debugging process if there are any errors present in the code.

3rd Test: First Iteration Test Results

Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73
Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73
Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73
Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73
Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73
Speed boost is on!	Player.js:65
Speed boost is off!	Player.js:73

[3rd Test: First Iteration Test Video](#)

The HUD feature is working correctly.

The test was successful.

Effects

Currently, when the player transforms into a monster; there's no transition for it, which looks quite dull and may negatively affect the user experience.

Therefore adding an effect when the player switches from monster to character would be a great way to add more visual transitioning of this special ability.

To add effects, I would recommend creating a class in the same file: **HUD.js**, this approach will follow a similar route of that of the HUD instances, where the HUD objects containing different HUD are passed as an attribute for the player class. In this case, using a class for adding effects will also give me the same benefits such as creating new effects easily by using one blueprint (effects class).

```
class FX{
    constructor(config){
        this.skin = new Image();
        this.name = config.name
        this.skin.src = "FX/" + config.name + "/SpriteSheet.png"
        this.isFinished = config.isFinished || false; // New property to control
the animation loop

        this.skin.onload = () => {
            this.isLoaded = true;
        }

        this.isFinished = false;

        this.animationsMap = config.animationsMap || {
            "play": [[1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0]]
        }

        this.animationSet = config.animationSet || "play";
        this.currentSpriteFrame = 0;
        this.framesLimit = 2;
        this.framesLeft = this.framesLimit;
    }
}
```

HUD.js

I have added a '**FX**' class to the **HUD.js** file, which is the blueprint or template for animation effects in the game. As seen in most classes in this project, I have taken config object as an argument which will contain attributes needed to create an instance of FX, this approach will allow me to configure different effects easily as I can create a FX object dictionary in **Global.js** and configure different effects that the player may use.

One major difference between other classes and the FX class is that I have added a new attribute called '**isFinished**' which is a boolean to indicate if the animation of the effect has iterated already or not. This will stop the effect from repeating itself over and over again even though the player is not toggling their transformation ability.

Moreover, I also added an **animationsMap**, similar to the walking animations for entities, the effects feature is going to have a spritesheet, therefore the game needs to draw the effects in a specific sequence and a specific frame at a time. Most of the effects in my asset that I am using follow the **animationsMap**, however, if an effect does have a different **animationsMap**, then I can configure the config object for that effect instance to have a specific **animationsMap** that it needs to use.

```

updateFramesLeft() {
    if (this.framesLeft > 0) {
        this.framesLeft--;
        return;
    }

    this.framesLeft = this.framesLimit;
    this.currentSpriteFrame += 1;

    if (this.animationsMap[this.animationSet][this.currentSpriteFrame]
        === undefined) {
        this.currentSpriteFrame = 0;
        this.isFinished = true; // Mark the animation as finished
        return;
    }
}

drawFX(context, camera) {
    const x = player.x + 9 * 16 - camera.x
    const y = player.y + 4 * 16 - camera.y
    // Only update the animation if it's not finished
    if (this.isLoaded && !this.isFinished) {
        const fx =
            this.animationsMap[this.animationSet][this.currentSpriteFrame][0] * 16
        const fy =
            this.animationsMap[this.animationSet][this.currentSpriteFrame][1] * 16
        context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
        this.updateFramesLeft()
    }
}
}

```

I added methods to the FX class, which are similar to the methods I added in the Sprite class, the main purpose for these methods is so that the game is able to draw each effect with the correct frame animation and at a correct speed.

The first method, **updateFramesLeft()**, will update the state of the effect animation frame by frame, by decrementing the remaining frames left to display correctly, I have implemented the same thing in the Sprite class.

The second method is the draw method, which will display the effect in the correct position and the correct frame.

I could use a library to manage animations such as TweenJs or GreenSock, these libraries are better than my original approach because it can simplify the and shorten the code which is required to manage the animation of the effect.

However, my approach is simpler and lightweight, making it easy to understand. I don't think using external libraries for medium-sized games like mine is really efficient.

Therefore, my approach allows the developers to have direct control over the animation state.

```
const playerEffects = {
  transformFX: new FX({
    name: 'Smoke',
  })
}

window.player = new Player({
  name: "MaskedNinja",
  transform: {
    name: "GoldRacoon",
    type: "monster"
  },
  x: 4, y: 4,
  hud: playerHUD,
  fx: playerEffects
});
```

Global.js

Similar to the HUD implementation, I have added a new object which stores graphic effects that the player will use. Currently, it only stores one effect so far, which is the **transformFX** which is only activated when the player uses their transform ability.

There are several other effects, but I decided to test it out with the "Smoke" sprite sheet.

Then, I pass in the playerEffects object into the player instance as an attribute.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the effect animation works	drawFX()	The effects animation should be drawn onto the correct position.
2	Check if the effect animation works when player transforms into a monster	drawFX() Player.update()	The effects animation should be displayed when the transformation ability is toggled on
3	Check if different effects works as well	playerEffects	The effects class should be capable of handling more and different effects with different sprite sheets

The main emphasis of this test plan is to check whether the game is able to display effects with the correct animation, correct position and be able to handle multiple different effects with different sprite sheets.

The first test is a **normal** test that determines whether the effect animation is drawn and iterated properly, which will help me to verify that the **drawFX()** works by confirming that the animation of the effects is drawn in the correct position, drawn with the correct frames and speed. I could skip this test to get through the development process faster, however, this test is really significant because it will help me to verify if the animations are drawn accurately onto the canvas which will provide important information about how to handle errors if there are any.

The second test is also a **normal** test which will ensure that this feature is mainly completed. It determines whether the effect animation works when the transformation ability is activated using the **Player.update()** method. This test is the most important test in this test plan. I could skip this test, however, it would be pointless as this test will determine if the development of this feature was successful.

The third final test in this test plan would also be a **normal** test that determines if the game and the effects class is able to handle different types of effects which consists of different sprite sheets. I could not test for this, however, this test would provide important information about the flexibility of this class, and the results will provide information about this class on how to increase its flexibility to handle multiple different types of effects, creating more variation in this game, rather than sticking to one single effect.

1st Test: First Iteration

The first test is to verify if the effect has the correct animation when being drawn onto the canvas and to verify if the effect is being drawn onto the correct position which is directly related to the player's position.

1st Test: First Iteration Test Results



[1st Test: First Iteration Test video](#)

The test was successful, the smoke effect is displayed in the correct position and in the correct animation.

The effect is also being iterated at a constant speed, I don't think I will need to configure the speed of this animation further as it seems to be already appropriate.

2nd Test: First Iteration

Second test, will allow me to know if I can combine the effects feature with the transformation ability feature for this game

```
class Player extends Obj { //GameObj that can be controlled by the user
  constructor(config) {
    super(config); //Inherits methods and attributes from Obj
    this.fx = config.fx
  }
}
```

Player.js

I have added the fx attribute to the player class so it can store the effects object passed into the player instance as an argument through the config objects.

```
class Player{
  //...
  update(state){
    //If there are no more tiles left to travel...
    if (this.tilesLeft === 0) {
      //if speedBoost is true and the player's speed is default
      if (state.speedBoost && this.speed === 1) {
        //...
        this.fx.transformFX.isFinished = false
        //Otherwise, if speedBoost is false and the player's speed is boosted
```

```

} else if (state.speedBoost === false && this.speed === 2) {
    //...
    this.fx.transformFX.isFinished = false
}
//...
}
//...
}
}
//...
}

```

Player.js

Now if the player uses their transformation ability, the transform effect will be toggled on by being marked as “**isFinished = false**”, this means that the game will need to iterate through the effect once.

2nd Test: First Iteration Test Results

[2nd Test: First Iteration Test Video](#)

The test was successful as the graphic transition effect is only activated when the player switches between monster and the default character.

3rd Test: First Iteration

```

const playerEffects = {
    transformFX: new FX({
        name: 'Shield',
    })
}

```

I have changed the transform effect from “**Smoke**” to “**Shield**”, to test if it can handle other effects from different sprite sheets.

3rd Test: First Iteration Test Results

[3rd Test: First Iteration Test Video](#)

The test was successful, the game can handle different transform effects. This also means that I have successfully met the seventh success criteria from my analysis section.

Success Criteria	Evidence	Justification
Effects & HUD	A video of the character using their special ability which will show an effect animation and display a HUD.	The seventh criteria, ensure that the game must have effects and HUD to provide important information or feedback during the game experience. I could not add effects and HUD to reduce visual clutter and maintain immersion. However, HUDs will allow the user to absorb essential information about the current state of the game, reducing confusion, and effects will make the game more enjoyable and engaging.

Character Transformation Review

I have added every feature needed for the special ability system in this game. Overall, the player is now able to switch between a character entity and a monster entity by holding the ShiftLeft Key, which will also toggle on the HUD that indicate whether the player is currently in monster mode or character mode, moreover, during the transition between these two entities, there is also a visual effect that takes place. I could have not added the additional HUD and effects feature to save time and simplify the project further, however, having a static transformation ability by itself may feel more boring to the user's gameplay experience.

Here's all the new code I have added during the development process of creating the character transformation ability, and with all the changes after any failed tests and refactoring.

```
class Sprite {
  constructor(config)
    //...
    this.name = this.Obj.name
    this.skin.src = "Characters/" + this.Obj.name + "/SpriteSheet.png"
    if(this.Obj.type === "monster"){
      this.skin.src = "Monsters/" + this.Obj.name + "/SpriteSheet.png"
    }
    //...
  }
  //...
  // This is a setter method for a property called "obj"
  set obj(obj) {
    // Set a private property called "_obj" to the given value
    this._obj = obj;
    // Set the "src" attribute using the name for the filepath
    this.skin.src = "Characters/" + this._obj.name + "/SpriteSheet.png";
    // If the object is a monster, change the filepath for monsters
    if(this._obj.type === "monster"){
      this.skin.src = "Monsters/" + this._obj.name + "/SpriteSheet.png";
    }
  }
  // This is a getter method for the "obj" property
  get obj() {
    // Return the value of the private property called "_obj"
    return this._obj;
  }
}
```

Entity.js

```

Class Obj {
  constructor(config){
    this.name = config.name
    this.type = config.type
    //Creates an attribute for the sprite or skin of the game object
    this.sprite = new Sprite({ Obj: this, animationSet: config.animationSet});
    //...
  }
  //...
}

```

Entity.js

```

class Player extends Obj {
  constructor(config) {
    // ...
    this.type = config.type || "character"
    this.behaviour = "standing";
    this.isPlayer = true;

    this.originalSprite = {
      name: this.name,
      type: this.type
    }
    this.transform = config.transform
    this.hud = config.hud
  }
  update(state){
    //...
    if (state.speedBoost && this.speed === 1) {
      //...
      This.sprite.obj = this.transform
      this.hud.transformHUD.opacity = 0.8
      this.fx.transformFX.isFinished = false
      //Otherwise...
    } else if (state.speedBoost === false && this.speed === 2) {
      //...
      This.sprite.obj = this.originalSprite
      this.hud.transformHUD.opacity = 0.3
      this.fx.transformFX.isFinished = false
    }
    //...
  }
}

```

Player.js

```

class HUD {
  constructor(config) {
    //The name will help to fetch the source
    this.name = config.name
    //The type of HUD of this instance
    this.type = config.type;
    // set default opacity to 1 if not provided
    this.opacity = config.opacity || 1;

  }

  // This function draws the Heads Up Display (HUD) for the game
  drawHUD(context) {
    // Check if the type of the object is a "transform" type
    if (this.type === "transform") {
      // Create a new Image object to hold the HUD image
      this.transformHUD = new Image();
      // Set the source path for the HUD image
      this.transformHUD.src = "HUD/NinePathRect/DialogueBubble2.png"
      // Create a new Image object
      this.image = new Image();
      // Set the source path for the HUD based on its name
      this.image.src = "Monsters/" + this.name + "/SpriteSheet.png"
      // Set the opacity canvas context to the given opacity
      context.globalAlpha = this.opacity;
      // Draw the HUD image on the canvas context at the specified coordinates
      context.drawImage(this.transformHUD, 8, 8)
      // Draw the HUD on the canvas context at the specified coordinates
      // and with the specified width and height
      context.drawImage(this.image, 0, 0, 16, 16, 16, 16, 16, 16)
      // Reset the opacity for the canvas context back to 1
      context.globalAlpha = 1;
    }
  }
}

```

HUD.js

```

class FX{
    // Constructor function that takes a config object as a parameter
    constructor(config){
        // Create a new image object called "skin"
        this.skin = new Image();
        // Set the "name" property to the value in the config object
        this.name = config.name
        // Set the "src" attribute of the "skin" object to a path
        this.skin.src = "FX/" + config.name + "/SpriteSheet.png"
        // Set a new property called "isFinished"
        this.isFinished = config.isFinished || false;
        // When the skin finishes Loading, set the "isLoaded" property to true
        this.skin.onload = () => {
            this.isLoaded = true;
        }
        this.isFinished = false; // Set "isFinished" to false again
        // Set a property called "animationsMap"
        this.animationsMap = config.animationsMap || {
            "play": [[1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0]]
        }
        // Set a property called "animationSet"
        this.animationSet = config.animationSet || "play"
        this.currentSpriteFrame = 0; // Set the current sprite frame to
        this.framesLimit = 2; // Set the frames limit to
        this.framesLeft = this.framesLimit; // Reset the frames left to draw
    }

    // Function to update the frames Left until the next sprite frame update
    updateFramesLeft() {
        // If there are still frames left until the next sprite frame update...
        if (this.framesLeft > 0) {
            //decrement the frames left and return
            this.framesLeft--;
            return;
        }
        // Reset the frames left to the frames limit
        this.framesLeft = this.framesLimit;
        // Move to the next sprite frame
        this.currentSpriteFrame += 1;
        // If there is no next sprite frame, reset to the first sprite frame
        if (this.animationsMap[this.animationSet][this.currentSpriteFrame]
            === undefined) {
            this.currentSpriteFrame = 0;
            this.isFinished = true; // Mark the animation as finished
            return;
        }
    }
}

```

```
// Function to draw the FX to the screen
drawFX(context, camera) {
    // Calculate the x and y coordinates for the FX
    const x = player.x + 9 * 16 - camera.x
    const y = player.y + 4 * 16 - camera.y
    // Only update the animation if it's not finished
    //the skin has finished loading
    if (this.isLoaded && !this.isFinished) {
        // Get the x and y coordinates for the current sprite frame
        const fx =
this.animationsMap[this.animationSet][this.currentSpriteFrame][0] * 16
        const fy =
this.animationsMap[this.animationSet][this.currentSpriteFrame][1] * 16
        // Draw the current sprite frame of the skin to the screen
        context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
        this.updateFramesLeft()
    }
}
}
```

HUD.js

```
Class Game{
    //...
    Loop() {
        //...
        //Draw the effects
        Object.values(player.fx).forEach(effect => {
            effect.drawFX(this.context, player)
        })
        //Draw Upper Layer
        this.map.drawUpper(this.context, player
        //Draw player HUD
        Object.values(player.hud).forEach(hud => {
            hud.drawHUD(this.context)
        })
        //...
    }
    //...
}
```

Game.js

Collisions

Collisions are the interactions that occur between the player and the objects , walls, and beings that come into touch with each other. It is an essential element because it allows the player to interact with the map's items. Without this crucial component, the game will be incomplete due to a lack of boundaries.

For example, a player entering a tile occupied by a wall or tree will be prevented from travelling further in that direction. Another type of collision might happen when the player interacts with other entities in the map, allowing the game to activate NPC dialogues or cutscenes. Without collisions, the player could move around anywhere they want which will remove the purpose of having maps in the game.

To summarise, collisions are really important in most games, especially top-down 2D games like mine, this feature will allow the player to interact and engage with the maps and entities in a meaningful way.

I could add overlapping bounding boxes instead of using pixel-perfect collisions. This approach can add boundaries between objects and the player by detecting when these two entities or objects intersect. Moreover, this approach might be better because it uses less resources and code to implement it, also it may increase performance too and will allow the player to pass through narrow gaps instead of being stopped for every pixel.

However, I am sticking to my original approach because of the tile based movement that I have implemented previously and pixel-perfect collision is easier to implement than overlapping bounding boxes, meaning my original approach will save me more time.

Collision Detection

```
{
  "compressionlevel": -1,
  "infinite": false,
  "layers": [
    {
      "data": [0, 0, 0, 1, 1, 1, 1, 1, 1, 603, 604, 601, 379, 0,
               0, 1, 600, 398, 0, 0, 0, 0, 491, 619, 620, 617, 389, 0,
               1, 597, 616, 0, 0, 0, 0, 0, 0, 0, 0, 0, 397, 379,
               1, 613, 0, 0, 0, 629, 647, 648, 649, 1, 0, 0, 0, 1,
               385, 631, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 389,
               1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 389,
               385, 613, 647, 648, 649, 645, 0, 0, 0, 0, 0, 0, 613, 387, 419,
               415, 418, 418, 418, 418, 418, 416, 0, 417, 418, 418, 418, 419, 0,
               0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
               0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      "height": 10,
      "width": 14,
      "id": 1,
      "name": "Walls",
      "opacity": 1,
      "x": 0,
      "y": 0
    }],
  }
}
```

Maps/StartingHouse/collision.json

To add collisions to my game, I will need data about the map which will indicate which tiles are free to travel to and which tiles are occupied by an object, wall etc.

The map editor that I am using allows me to export maps in the json format which also includes data about free tiles, height and width. This data can help to add collisions.

I could add collisions by adding one coordinate at a time where the player is able to travel to, this alternate approach is really simplistic and easy to understand.

However, since I am going to add a few maps, one of them being really large, this procedural approach will take me a really long time to add collisions. My approach will use the collision data to create collisions using OOP, this approach is more efficient, saves a lot of time and resources, shortens the code, and facilitates adding new maps.

```

class Map {

    //...

    // Function to check for collision between player and walls
    checkCollision(){
        // Check if player is within map bounds
        if(player.x >= 0 && player.x < this.walls.width*16 &&
            player.y >= 0 && player.y < this.walls.height*16){
            // Check if there are any walls on the current map instance
            if (Object.keys(this.walls).length !== 0){
                // Check if player is colliding with a wall using collision data
                if(this.walls.data[player.x + (player.y * this.walls.width)] ==
                    === 0) {
                    console.log('free space')
                } else{
                    console.log("collision")
                }
            }
        }
    }

    // Function to fetch collision data for the current map instance
    fetchCoordinates(){
        // Fetch collision data for the current map instance
        fetch('/Maps/' +this.name+ '/collision.json')
        // Store collision data in this.walls
        .then(response => response.json())
        .then(json => {
            this.walls = json.layers[0]
        })
    }
}

```

I have added two new methods in the Map class after drawing the map layers.

The first method, `checkCollision()`, checks for collisions between the player and walls on the map. The second function, `fetchCoordinates()`, fetches the collision data for the current map instance from a JSON file.

The `checkCollision()` method first checks whether the player is within the boundaries of the map. It then checks whether there are any walls on the current map instance using the collision. Finally, it checks whether the player is colliding with a wall using the collision data.

The `fetchCoordinates()` method fetches collision data for the current map instance from a JSON file. This collision data is then stored in the `walls` attribute of the `Map` instance.

These two methods provide an effective way to check for collisions in a game map. I could include using a physics engine to handle collisions or manually checking for collisions between all game objects. However, using a physics engine may be unnecessary for a simple game and could add unnecessary complexity. Manually checking for collisions between all game objects could also be inefficient and may not be scalable as the number of game objects increases.

Overall, the approach taken in the `Map` class strikes a balance between simplicity and efficiency. It allows for effective collision detection without adding unnecessary complexity or sacrificing performance.

```
//Game Loop
Loop() { //Loop method for the game
  const gameLoop = () => { //Declares a game Loop function
    //Calls a function before going repainting the next frame
    requestAnimationFrame(() => {
      //...
      this.map.checkCollision();
      //...
    })
  }
}
```

Game.js

To run this function, I have called the `checkCollision()` function in the game loop so the game is constantly checking for any collisions.

```
//...
init() {
  this.map = new Map(window.Maps.StartingTown)
  this.map.fetchCoordinates()
  //...
}
```

Game.js

I have configured the `init()` method of the `Game` class; I have changed the initial map to `StartingTown` because it would be more suitable for the test as it is bigger in size compared to the `StartingHouse` map.

The `fetchCoordinates()` function is then called to retrieve the map data for the map that is being initialised, in this case the map data of `StartingTown` is being retrieved.

Test

Test No.	What I am testing	Test Code	Result Required
1	Check if the game displays if the player stands on an occupied tile	checkCollision()	If the player stands on an occupied tile then print "collision". Or else print "free space".
2	Check if the game displays if the player stands walks collides with an entity	checkCollision()	If the player stands on a tile occupied by an entity, then print "collision". Or else print "free space"

The first test is a **boundary** test which determines whether the game is capable of checking if the player stands on a tile that is occupied by a map wall, and should print "collision" or "free space" in the console accordingly. This test will help me verify if the camera is ready or capable of handling the development of the actual collision, as checking whether the entities are colliding is the first big stepping stone for creating actual working collisions in the game. The second test is also a boundary test which determines whether the game is capable of checking if the player is colliding with an entity or not. This is similar to the first test, but with the only expectation being that the game is checking collision between entity objects rather than map walls. I could combine both of these tests together to speed up the development and testing phase of this project and keep on track with the schedule and gantt chart, however, these tests are focused on individual specific functionality of the **checkCollision()** method, meaning that it would be more appropriate to take my time in detecting if the **checkCollision()** method is works for which. Furthermore, if there are any errors, It would be easily detectable if I separate the two tests because the error would likely affect only one aspect of the **checkCollision()** method rather than both the entity collision and map collision.

1st Test: First Iteration

This test is going to show if the game can detect whether the player is standing on an occupied tile. If the tile is occupied, the game should print out "collision" in the log, otherwise, if the tile is free to travel on, the game should print out "free space".

```
42 collision
  Speed boost is on!
32 collision
24 free space
  Speed boost is off!
2 free space
```

[1st Test: First Iteration Test](#) [1st Test: First Iteration Test Video](#)

The test was successful as the game can check whether the player is colliding or not, this will now facilitate when I add working collisions in the game.

2nd Test: First Iteration

The second test is like the first test, however instead of testing walls and map objects, I am testing collisions with entities.

```
// Function to check for collision between player and walls
checkCollision(){
    // Check if player is within map bounds
    if(player.x >= 0 && player.x < this.walls.width*16 &&
        player.y >= 0 && player.y < this.walls.height*16){
        // Check if there are any walls on the current map instance
        if (Object.keys(this.walls).length !== 0){
            // Check if player is colliding with a wall using collision data
            if(this.walls.data[player.x + (player.y * this.walls.width)] === 0) {
                console.log('free space')
            } else{
                console.log("collision")
            }
        }
    }

    Object.values(this.entities).forEach(entity => {
        if(player.x/16 == entity.x/16 && player.y/16 == entity.y/16){
            console.log("colliding with entity")
        }
    })
}
```

2nd Test: First Iteration Test Results

[2nd Test: First Iteration Test Results](#)

colliding with entity	Map.js:51
free space	Map.js:41
colliding with entity	Map.js:51
free space	Map.js:41
colliding with entity	Map.js:51
free space	Map.js:41
colliding with entity	Map.js:51

The test was not entirely successful, when the player intersected with the entity, the game considered that tile as a free space and collision with the entity. This could cause some errors in the future when I add working collisions, therefore this is a major flaw.

The test was partially successful.

2nd Test: Second Iteration

```
// Function to check for collision between player and walls
checkCollision(){
    let collision = false
    // Check if player is within map bounds
    if(player.x >= 0 && player.x < this.walls.width*16 &&
        player.y >= 0 && player.y < this.walls.height*16){
        // Check if there are any walls on the current map instance
        if (Object.keys(this.walls).length !== 0){
            // Check if player is colliding with a wall using collision
            // data
            if(this.walls.data[player.x + (player.y * this.walls.width)] ==
                === 0) {
                collision = false
            } else{
                collision = true
            }
        }
    }

    Object.values(this.entities).forEach(entity => {
        if(player.x/16 == entity.x/16 && player.y/16 == entity.y/16
            && !entity.isPlaying){
            collision = true
        }
    })
}

console.log(collision)
}
```

I could not use this new approach, however, the second approach is better than the first one for several reasons. First, it uses a variable to keep track of the collision status instead of printing to the console directly. This makes it easier to use the function for other purposes, such as triggering an event or changing the game state. Moreover, the second approach is more efficient since it does not print to the console multiple times. Instead, it prints only once after all the collision checks are performed. Finally, this new approach is more readable and easier to understand since it separates the collision checking logic from the console output.

2nd Test: Second Iteration Test Results

2nd Test: Second Iteration Test Video

The test was successful, the game can now detect whether the player is colliding with a tile which is occupied by a wall or an entity.

Working Collisions

Now I can add working collision, where the player will be stopped if they tried to travel into an occupied tile, this is going to use the collision detection feature that we just implemented.

```

Loop() { //Loop method for the game
    const gameLoop = () => { //Declares a game Loop function
        //Calls a function before going repainting the next frame
        requestAnimationFrame(() => {
            //...
            this.map.checkCollision()
            // Iterate through each entity in the list
            entities.forEach(entity => {
                // Initial state
                const state = { map: this.map };
                // add the player's direction and speedBoost to the state
                if (entity.isPlaying) {
                    state.direction = this.keyInput.direction;
                    state.speedBoost = this.keyInput.speedBoost;
                }
                // passing in the state for every other entity
                entity.update(state);
            });
            //...
        })
    }
}

```

Game.js

Now that I have passed in the current map instance into the entities and player, I can proceed to implementing logic in the player update method. My approach is to set **this.tilesLeft** to 0 if the player runs into a collision, this will prevent the player from travelling. I could have kept the old approach of calling the **checkCollision()** function directly from the game loop which would have reduced the length of the code and won't require the map instance to be passed on to each entity object, which could improve efficiency and performance as this map method will be code once in every loop iteration rather being called for every entity presented in the current map in one game loop iteration. However, passing in the map instance as a state to each entity allows for more flexibility and the map will have access to the entity instance directly, meaning that I won't need to implement a separate method for the map class to provide collision for each entity, because if you want to collide two objects, the game needs to set their **this.tilesLeft** to 0. Therefore, the alternative approach is unable to do so, as a result, putting the map instance attribute in the state object and passing this state to the **entity.update()** method is far better than the older approach.

```

class Player extends Obj {
    //...
    update(state){
        //...
        //Updates player's direction when TilesLeft is 0
        if (this.TilesLeft === 0 && state.direction) {
            this.direction = state.direction;
            //if there is a collision...
            if (state.map.checkCollision(this)) {
                //Don't let the player travel further
                this.TilesLeft = 0;
                this.behaviour = "standing"
            } else {
                //Otherwise, let the player move.
                this.TilesLeft = this.speed * 16;
                this.behaviour = "walking";
            }
        }
    }

    //Updates the player's sprite
    updateSprite(state) {
        if (this.TilesLeft === 0 && !state.direction) {
            //...Then set the player's animation to 'idle' + direction
            this.sprite.updateSpriteSet("idle-" + this.direction)
            this.behaviour = "standing";
        } else {
            if (this.behaviour = "walking") {
                //...Otherwise, set the player's animation to 'walk' + direction
                this.sprite.updateSpriteSet("walk-" + this.direction)
            }
        }
    }
}

```

Player.js

I have added a few changes to the player class, especially to the **update()** method, where I have added a few conditional statements. The first statement checks if the player does not have tiles left to travel and has received a key input for its movement feature, if so, the player's direction is updated accordingly and the function now checks if there are any collisions in front of the player, by calling the **checkCollision()** passed in in the state object. If this method returns true, the player will be prevented from moving any further to that direction, otherwise the player will be able to travel as normal. I have executed this logic by utilising the binary behaviour attributes which store either “standing” or “walking”. These flags then help the other key movement functions such as the **updatePos()** method to figure out if the player can move or not, as I have explained previously.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the player stops if there is a collision	checkCollision()	The player should not be able to travel to a tile where it is occupied by anything such as entities, walls and other objects.
2	Check if the collisions works in different maps	checkCollision()	The collisions should work in other maps as well as every map has the same structure for the collision data in their json file.

The purpose of this test plan is if the collision detection system is able to determine if there are any collisions between the player and everything else such as map walls, entities etc.

If both of these tests are completed and show successful results are described in the result required column, then there will be complete working collisions in the game meaning that I have met one of the criterias in my success criteria. These tests will allow me to verify the functionality of the **checkCollision()** function and verify if it is still effective in other maps with different collision data.

1st Test: First Iteration

1st Test: First Iteration Test Results



[1st Test: First Iteration Test Video](#)

The test was unsuccessful as the player seems to stop however it's stopping at the wrong tile.

Error:

The game only activates the collision when the player is on the occupied tile instead of activating the collision before the player moves into the occupied tile.

Solution:

The **checkCollision()** functions should take into account the direction of the input key and check if the tile that the player is travelling to is occupied or not. If it is, then set TilesLeft to 0.

1st Test: Second Iteration

```

checkCollision(player) {
    // Set initial value of collision as false
    let collide = false;

    // Define a dictionary of direction with corresponding x and y values
    let directionDict = {
        "up": [0, -1], "down": [0, 1],
        "right": [1, 0], "left": [-1, 0]
    };

    // Destructure x and y values from the dictionary based on the input direction
    const [xValue, yValue] = directionDict[player.direction];

    // Update x and y values based on direction
    const x = Math.round(player.x / 16 + xValue);
    const y = Math.round(player.y / 16 + yValue);

    // Check if new x and y values are within the bounds of the walls
    if (x >= 0 && x < this.walls.width && y >= 0 && y < this.walls.height) {
        // Check if there are any walls defined
        if (Object.keys(this.walls).length !== 0) {
            // Check if the current position contains a wall
            if (this.walls.data[x + (y * this.walls.width)] !== 0) {
                collide = true;
            }
        }
    };
}

Object.values(this.entities).forEach(entity => {
    if(x == entity.x/16 && y == entity.y/16 && !entity.isPlaying){
        collide = true
    }
})

return collide;
}

```

I have modified the code for the solution. I have added a dictionary, **directionDict**, which I have implemented before. This variable matches the x and y values with the direction specifying the value to add to the entity's position based on the axis. For example, if the direction is up, then **0** would be added to the entity's x position (meaning there is no change) and **-1** would be added to the entity's y position, which means that the entity's y coordinates are being subtracted as they go up. To fetch these values, I have destructured the **xValue** and **yValue** based on the entity's current direction. Then, I decided to add these values with the player position

coordinates to create new x and y variables and rounded to the nearest integer. These two new x and y variables are the coordinates which the player is about to move to. The function then checks if that coordinate is free to travel or not using the JSON file and a few conditional statements, determining whether the player can move one tile in the direction they are facing. This code should essentially fix the previous errors and be used to handle entity movement collision more effectively.

1st Test: Second Iteration Test Results

[1st Test: Second Iteration Video Test](#)

The test was successful as the player collides with walls and entities

2nd Test: First Iteration

In the second test, I am testing if the collisions work in other maps as well. I have switched the map to the **StartingHouse** which also obviously has its own **collision.js** with collision data.



2nd Test: First Iteration Test Results

[2nd Test: First Iteration Video Test](#)

The test was successful as the player collides with walls and entities in different maps.

I have also now met the the eight success criteria

Refactoring

Now that the collisions work, I can now do some refactoring and increase their performance:

```
async fetchCoordinates(){
    // Make a fetch request to retrieve collision data for the map
    const response = await fetch(`/Maps/${this.name}/collision.json`);
    // Parse the response as a JSON object
    const json = await response.json();
    // Assign the first layer of the JSON object to the "walls" property
    this.walls = json.layers[0];
}
```

I have refactored the `fetchCoordinates()` method to use asynchronous programming with `async` and `await` keywords. The `fetch` request has been updated to use template literals, making the code easier to read. The `.then` chains have been replaced with `await` to wait for promises to be resolved before moving on to the next line of code. The first layer of the JSON object is assigned to the `walls` property after the response has been fully processed.

Map Navigation

In my game, it is necessary to add map switching, where the player can travel between maps by entering and exiting maps when they reach a certain tile. This will allow the user to explore the different maps. These maps will contain different objects, purposes and entities that the player can interact with.

I could just add every single feature in one big map, this approach is easier as my code does not need to handle different maps etc. However, this approach may negatively affect user experience and limit gameplay time, as a result, the user may feel the game has a lot of things to do in this game. Adding different maps will incentivise the player to explore the game further and will increase their gameplay time.

```
window.Maps = {
  StartingHouse: {
    name: 'StartingHouse',
    //...
    exits: {
      StartingTown: {
        name: "StartingTown",
        x: 7, y: 7,
        newX: 28, newY: 9
      },
    },
  },
  //...
}

//Creates instances of the maps provided in window.Maps.
window.mapDict = {
  StartingHouse: new Map(window.Maps.StartingHouse),
  StartingTown: new Map(window.Maps.StartingTown),
};
```

Global.js

I have added a new attribute called **exits** to StartingHouse in the window.Maps. The **exits** objects contain the maps where the current map can travel to. For example, the **StartingHouse** only has one map that it can exit to, which is **StartingTown**.

The maps that the player can travel to are given a few attributes; the name of the map, the x and y coordinates where the player needs to stand to switch to that map, and the new x and y coordinates that the player will teleport to in the new map. I have added a new global object which initialises map instances provided in **window.maps** rather than initialising a map instance in the **init()** method of the Game class in Game.js.

This new approach is better because the map instances are already initialised and loaded before the game starts so the player can move into maps that already exist rather than creating a new instance every time the player switches a map.

This new approach will further facilitate map switching as I can change this.map attribute of the game class to 'mapDict[StartingHouse]' for example, which will allow me to switch maps instantly. I have implemented this change in the Game class's init() method as seen in the following code.

```
Class Game{
    Loop{
        //...
        //For each exit in the current map...
        Object.entries(this.map.exits).forEach(([key, exit]) => {
            //If the player coords matches the exit coords to travel to that map...
            if (player.x / 16 === exit.x && player.y / 16 === exit.y) {
                //Fetch the instance of that new map and store it to newMap
                const newMap = Object.keys(window.mapDict).find(k =>
                    window.mapDict[k].name === exit.name);
                //Switch to the new map
                this.map = window.mapDict[newMap];
                //Fetch the coordinates from the new map's collision.json
                this.map.fetchCoordinates();
            }
        });
        //...
    }

    //The init method will start the game
    init() {
        this.map = mapDict.StartingHouse
        this.map.fetchCoordinates()
        //...
    }
}
```

Game.js

In the following code, I have added some new logic to the loop method of the game class. The purpose of this additional code is to constantly check if the player has toggled map transition by checking if the player is standing only an 'exit tile' meaning that travelling to that tile should activate map switching. The code ensures if the player is standing on a 'exit tile' for every single exit map in for the current map.

If the player does stand on an 'exit tile', the code should prepare to switch maps. Therefore, a new variable called **newMap** is created which stores the instance of the new map, to do so, the code checks the map.name attribute of each map instance in **window.mapDict**, if one of the

map.name attribute matches **exit.name**, then the game will make sure to change to the new map. When the instance of the new map that the player wants to travel to is fetched, the game can now change maps safely, the **this.map** attribute is changed to the instance of the new map using the **window.mapDict**, and the **fetchCoordinates()** method is called so that the game can access collision data and set working collisions for the new map.

In my approach, I have used an event-based system to handle map transition. I could use a different approach; the game can listen for an event which is triggered when the player steps on an 'exit tile', rather than checking if the player has stepped on an 'exit tile' in every loop iteration. The alternative approach is more efficient because it doesn't constantly check for 'exit tiles', however, this approach may lead to the game missing a map switch trigger if the player moves too quickly over the 'exit tile'. Therefore, my original approach is better for handling map transitions because it is safer even though it is less efficient.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the player is able to switch maps	this.map.exitsLoop()	The player should be able to switch maps by standing on an exit tile of the current map and teleported to the new map with the correct position
2	Check if the player is able to travel between more than two maps.	this.map.exitsLoop()	The player should be able to travel and access between the following maps: StartingHouse, StartingTown and another map called 'Dojo'

The first test is a **normal** test which determines whether the game is capable of handling map switching by teleporting the player to the desired map when entering an 'exit' tile. The player should have the correct position and teleport to the correct map. This test is significant especially to ensure that the game's navigation system is developed properly and efficiently, which is a major requirement to meet the success criteria of my project.

The second test is a **boundary** test that determines whether the game allows the player to navigate between three different maps in one setting. This will ensure that the development of this feature is capable of handling multiple maps allowing the game to handle complex map navigation. The main purpose of this test is to detect any potential errors and to solve these issues that it may come with.

1st Test: First Iteration



[1st Test: First Iteration Test Results](#)

[1st Test: First Iteration Test Video](#)

The test was unsuccessful because the player did not get teleported to the correct position when it switched from **StartingHouse** to **StartingTown**

Reason:

I have not updated the player's coordinates to the new coordinates when it switches to the new map, which could be the reason why the player's coordinates were not changing when the player navigated to a different map.

Solution:

Add a piece of logic where the player's coordinates are updated when the player switches between the different maps.

1st Test: Second Iteration

```
Class Game{
    Loop{
        //...
        //For each exit in the current map...
        Object.entries(this.map.exits).forEach(([key, exit]) => {
            //If the player coords matches the exit coords to travel to that map...
            if (player.x / 16 === exit.x && player.y / 16 === exit.y) {
                player.x = exit.newX*16;
                player.y = exit.newY*16;
                //Fetch the instance of that new map and store it to newMap
                const newMap = Object.keys(window.mapDict).find(k =>
                    window.mapDict[k].name === exit.name);
                //Switch to the new map
                this.map = window.mapDict[newMap];
                //Fetch the coordinates from the new map's collision.json
                this.map.fetchCoordinates();
            }
        });
        //...
    }
}
```

I have updated the player coordinates when map switching is activated. It fetches the newX and newY coordinates and sets them to the player's coordinates when it switches the maps.

1st Test: Second Iteration Test Results[1st Test: Second Iteration Test](#)

The test was successful, and the new addition has fixed my previous error as expected.

2nd Test: First Iteration

```
window.Maps = {
  ...
  Dojo: {
    name: 'Dojo',
    entities: {
      ...
    }
    exits: {
      map1: {
        name: "StartingTown",
        x: 8, y: 17,
        newX: 36, newY: 16
      },
    }
  }
}
```

Global.js

I have added a new map called '**Dojo**' which can only be accessed from **StartingTown** at the coordinates [8,17].

This is a new addition to my game as the player will have more places and maps to travel to, extending the gameplay for the user and enhancing user experience as they can explore different maps in the game.

Furthermore, this new addition will also help me with the following test, where I am testing if the player can travel between more than 2 maps.

[2nd Test: First Iteration Video](#)

The test was successful, the player is able to travel between three maps in one setting. I have added map switching to the game successfully. I have also successfully met the tenth success criteria from the analysis stage.

Success Criteria	Evidence	Justification
Map Switching	Video of the player exploring and switching between different maps while displaying a transition.	The tenth criteria ensures the game must have a map switching mechanic to allow players to discover and explore new maps. I could not add this feature as it may make development more confusing, however, if there are no map switching features, there is no point of having multiple different maps as the user cannot have a system to access and travel to them, restricting exploration.

Map Transition

The map transition feature will add a graphical transition when the player switches maps. I am planning to consult with my stakeholder in this matter about this feature and the details that I can implement in this transition feature. This feature was not mentioned before for the plan for developing this game, however, I think this is necessary at this stage because when the player travels between maps, there is a lack of transitioning and the maps are switching too fast. Therefore adding a transition will enhance immersion because of the smooth transitioning, this allows the player to deduce that they are entering a new map. It will also reduce confusion for the player if they accidentally walk on an exit tile, if there is no transition process, it may disorient the player because they may have switched maps too quickly.

Stakeholder Consultation

After consulting with one of my stakeholders who specialises in graphic design, Muhaimin Ahmed, has advised for the following criterias that may make this feature more appealing to the nature of this game and enhance map navigation and the exploration aspects of this game.

Interview

What type of transition should be the best in the context of my game?

Muhaimin: "I believe that a fade-in screen from displaying the map on the canvas fading into a solid colour then to the new map should be a really easy method to implement and it is also really common in most RPG games similar to yours, this approach would be the best in my opinion"

What solid colour should I configure the transition with?

Muhaimin: "Black will be the best suitable colour in this case as it provides a sense of mystery when the player enters a new map and doesn't know what to expect, increasing the immersiveness of the game".

For how long should the transition carry out for?

Muhaimin: "1 second or less should be the ideal time because you do not want to make the game feel slow, it is better to keep the speed fast and easily configurable if needed because different users may have different preferences."

I could not add this feature, which will save more time and will require less modification to the code, which will maintain the simplicity of this project. However, this addition will make the game more seamless, more engaging and allow for faster loading time for fetching collisions, map layers, and other map data which will prevent errors and bugs in the game as the maps will seem already loaded once the player finishes the transition process. Moreover, map transitions are a common practice in most games, especially 2D RPG games such as mine.

```

Loop() {
    let fadeIn = false;
    let fadeInOpacity = 1;
    const gameLoop = () => {
        if (!fadeIn) {
            //...

            // Check for player exiting or entering a new map
            Object.entries(this.map.exits).forEach(([key, exit]) => {
                if (player.x / 16 === exit.x && player.y / 16 === exit.y) {
                    // Start the fade in transition
                    fadeIn = true;
                    fadeInOpacity = 0;
                    this.map = window.mapDict[this.map.update(key, exit)];
                    this.map.fetchCoordinates();
                }
            });
        }

        //...
    }

    if (fadeIn) {
        // If fade in is true, fill the canvas with black and increase opacity
        this.context.fillStyle = "black";
        this.context.globalAlpha = fadeInOpacity;
        this.context.fillRect(0, 0, this.canvas.width, this.canvas.height);

        fadeInOpacity += 0.1;
        if (fadeInOpacity >= 1) {
            // If opacity has reached 1, stop the fade in
            fadeIn = false;
            this.context.globalAlpha = 1;
        }
    }

    requestAnimationFrame(gameLoop);
};

gameLoop();
}

```

Game.js

I have added the fade-in feature into the game loop in Game.js; where I have declared a few variables, setting **fadeIn** as false. This boolean will let the game know if to activate the transition process or not. The second variable, **fadeInOpacity**, as 1.

When `fadeInOpacity` is 1, it means that the black screen is not visible.

When `fadeInOpacity` is 0.5, it means that the black screen is partially visible.

When `fadeInOpacity` is 0, it means that the black screen is totally visible and the whole canvas is covered in black. The approach of adding this feature is to pause the game and gradually increment the `fadeInOpacity` when the player steps on an 'exit tile', when the map switch is completed, the `fadeInOpacity` is set back to 0. This creates a fade-in visual effect.

In order to achieve this effect, I have set `fadeIn` = true and `fadeInOpacity` = 0, when the player steps on an exit tile. This signals the game to activate the transition process.

When the drawing methods are finished iterating, the code checks if the `fadeIn` variable is true, if so, it will activate the transition process; The game loop then enters a `fadeIn` block, where the canvas is filled with a black colour using the `fillStyle` property.

The opacity of the canvas is set to `fadeInOpacity` using the `globalAlpha` property of the context object.

The `fadeInOpacity` variable is incremented by 0.1 in each iteration of the loop until it reaches a value of 1 (Fully-black canvas). Once the opacity reaches 1, the `fadeIn` variable is set to false, meaning that the transition effect is complete, and the `globalAlpha` property is set back to 1. In conclusion, this provides a smooth transition.

I could use an alternative approach such as displaying a simple video overlay on top of the canvas, then fades in or fades out, and only activated only when the player navigates to a different map. This alternative approach is better for a multitude of reasons as it is easier to implement and does not require much code, but instead can be executed by a one-liner. However, my approach is better because the alternative approach will be less flexible because I won't be able to configure the attributes of a video, whereas my approach of implementing a fade-in feature within the game loop is easily configurable.

Test

Test No.	What I am testing	Test Code	Result Required
1	Check if the canvas fades from the current map to the black screen	<code>Loop()</code>	The screen should fade from current map to black screen then to the new map while pausing the game temporarily
2	Check if the canvas <code>fadeIn</code> transition speed can be configured	<code>Loop(), fadeInOpacity</code>	The screen should fade-in at a different speed, faster or lower.

The first test is a **normal** test which determines whether the canvas is capable of displaying a black screen fade-in effect when the player navigates to a different map.

This test is important because it is the main purpose and the main functionality of this feature

The second test is a **boundary** test which determines whether the fadeln transition speed can be configured correctly and free of errors. I am testing with extreme values specifically for this test. This test is also important because it will give me more flexibility to configure the speed of the fadeln transition therefore I can customise the transition based on what the stakeholders and users will find more engaging or immersive.

1st Test: First Iteration

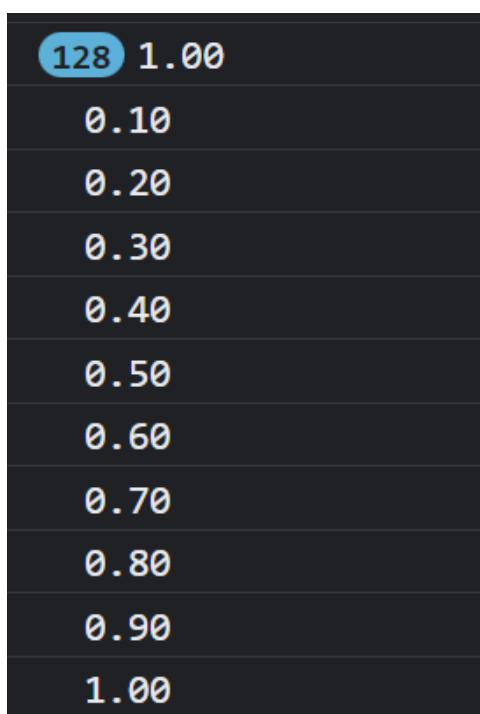
```
Loop() {
    let fadeIn = false;
    let fadeInOpacity = 1;

    const gameLoop = () => {
        console.log(fadeInOpacity.toFixed(2));
        if (!fadeIn) {
            //...
        }
    }

    gameLoop();
}
```

In this code, I have added a console function so I can keep track of the opacity of the transition screen when it is activated and going through the fadeln process.

1st Test: First Iteration Test Results



[1st Test: First Iteration Test Video](#)

The test was successful, the game pauses and displays the fade-in transition effect when the player activates map switch, allowing the player to navigate through different maps.

2nd Test: First Iteration

```

Loop(){
    //...
    if (fadeIn) {
        // If fade in is true, fill the canvas with black and increase opacity
        this.context.fillStyle = "black";
        this.context.globalAlpha = fadeInOpacity;
        this.context.fillRect(0, 0, this.canvas.width, this.canvas.height);

        fadeInOpacity += 0.05;
        if (fadeInOpacity >= 1) {
            // If opacity has reached 1, stop the fade in
            fadeIn = false;
            this.context.globalAlpha = 1;
        }
    }
    //...
}

```

In the game loop, the '**fadeInOpacity**' variable is incremented by **0.05** for every iteration of the game loop. This configuration will allow the fadeIn process to be slower, this is because **0.05** is less than **0.1**, meaning that it takes more iterations to reach 1 if the fadeInOpacity is increased by **0.05**, rather than **0.1**.

This will allow me to test if the speed of the fadeIn transition will change accordingly.

2nd Test: First Iteration Test Results

110	1.00
0.05	
0.10	
0.15	
0.20	
0.25	
0.30	
0.35	
0.40	
0.45	
0.50	
0.55	

[2nd Test: First Iteration Video](#)

The test was successful. The fade-in effect seems to be slower than the normal speed of the fade-in effect.

I have successfully added the map switching feature alongside with the map transition. It is time to review this major feature of this game.

Map Switching Review

Here's the major changes that I have added for this feature.

```

class Game{ //...
  Loop() {
    //Initially, there should be no fade transition
    let fadeIn = false;
    let fadeInOpacity = 0;
    const gameLoop = () => {
      if (!fadeIn) {
        //...
        // Check for player exiting or entering a new map
        Object.entries(this.map.exits).forEach(([key, exit]) => {
          if (player.x / 16 === exit.x && player.y / 16 === exit.y) {
            // Start the fade in transition
            fadeIn = true;
            fadeInOpacity = 0;
            //Change to the respective map
            this.map = window.mapDict[this.map.update(key, exit)];
            //Traverse the collision json file for the new map
            this.map.fetchCoordinates();
          }
        });
        //...
      }
      // If fade in is true
      if (fadeIn) {
        //The fade-in transition colour
        this.context.fillStyle = "black";
        //The Level of opacity
        this.context.globalAlpha = fadeInOpacity;
        //Ensures the transition covers the whole canvas
        this.context.fillRect(0, 0, this.canvas.width, this.canvas.height)
        //Keep increasing the opacity of the transition
        fadeInOpacity += 0.025;
        if (fadeInOpacity >= 0.6) {
          // If opacity has reached the opacity limit, stop the fade in
          fadeIn = false;
          this.context.globalAlpha = 1;
        }
      }
      requestAnimationFrame(gameLoop);
    };
    gameLoop();
  }
}

```

Game.js

NPC Behaviour

Movement and Collisions

```
//A blueprint for an object in the game
class Obj {
    //...
    //The Update method will commit changes to the object
    update(state) {
        //Call the entity movement methods
        this.updatePos()
        this.updateSprite()

        //If the entity has tiles left to travel...
        // and If there is a collision tile in front of the player
        if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16)
            && state.map.checkCollision(this.x/16, this.y/16,
this.direction)){
            //Stop the entity by updating the TilesLeft and behaviour
            this.TilesLeft = 0;
            this.behaviour = "standing"
            //Otherwise, let the entity keep travelling
            } else{
                this.TilesLeft = this.speed * 16;
                this.behaviour = "walking";
            }
        }
    }
    //...
}
```

Entity.js

I have added an **update()** method, similar to the player class, ensures that the entity's state is updated for each iteration in the loop, such as their position, sprite, and other features like checking for any collisions and movement which I am going to add in this development stage. This method allows the entity's state to be updated by calling other methods within itself such as the updatePos() and updateSprite() method, which will store similar logic to the player's classes methods, however, these methods will be a little more different to accommodate for future development of new features for NPCs and monster entities such as behaviour loops and interactions.

Furthermore, this method checks if the entity is on a tile within the grid and has any tiles left to travel. If it does, then the code checks for any collision in the next tile that the entity about to travel using the **checkCollision()** method I developed in the previous stages, this function will return a boolean, true meaning that the entity is colliding and false meaning that the entity is free to travel to the next tile. If the entity is colliding, meaning that the **checkCollision()** function has returned true, then the entity's **this.tilesLeft** attribute will be set to 0 and their behaviour will be set to 'standing', preventing the entity to travel any further, creating a collision. If the entity is not colliding, then the entity will be allowed to travel, taking in account their speed when setting the **this.tilesLeft** attribute, and the behaviour will be set to 'walking'.

This logic is run every time the entity instance of this class is on top of a tile within the grid, meaning that their position is not a decimal but rather an integer as they are standing or walking on top of a complete tile. This approach is really effective, easy to implement, allowing the game to handle multiple entities at once thanks to the use of OOP, the main purpose of this code is to prevent other entities, which are not controlled by the user, such as monsters and NPCs to collide with map objects like walls etc.

I could use an alternative implementation such as allowing the entity to have a velocity attribute that determines how it moves each frame; this would also require a different method for collision, which will be required to detect and handle collisions between the entity and other map objects. This approach may be better because it provides more control over the entity's movement, is more flexible and separates collision handling into a single different method which both detects and handles collisions for the entity which is more efficient than my original approach. However, this alternative approach is more complex as it requires more code to calculate the entity's velocity attributes and will make the code more difficult to understand for developers who are not familiar with collisions and tile-based movement, which could affect the maintenance in future development. Therefore, I am more inclined to stick to my original approach as not only is it less complex and more simple, but also makes it easier for other developers to understand and update the code for better maintenance.

```

updatePos() {
    if (this.tilesLeft > 0) { //If the entity has to move
        //Checks which direction it needs to move to
        const [axis, value] = this.directionDict[this.direction]
        //Changes their position value on the correct axis
        this[axis] += value * this.speed
        // If this method continues to run, the method will stop when the
        tilesLeft is 0
        this.tilesLeft -= this.speed * this.speed;
    }
}

```

```

//Updates the player's sprite
updateSprite() {
    if (this.tilesLeft === 0) {
        //...Then set the player's sprite animation to 'idle' + direction
        this.sprite.updateSpriteSet("idle-" + this.direction)
        this.behaviour = "standing";
    } else {
        if (this.behaviour = "walking") {
            //...Otherwise, set the player's sprite animation to 'walk' +
            direction
            this.sprite.updateSpriteSet("walk-" + this.direction)
        }
    }
}

```

Entity.js

The previous two methods, **updatePos()** and **updateSprite()**, I have already coded it in the development of the player class tiled-based movement and the animation movement, as a result, I can delete those methods from the player class and instead move into the entity class, so both classes can access it, as the player class already inherits all the attributes and methods from the entity class. I could keep the code as it was, by adding two separate methods (**updatePos()** and **updateSprite()**) to two separate classes (**Player Class and Entity Class**) which will increase the flexibility of the code. However, this would remove the main purpose of the player class inheriting from the entity class and may lead to inefficient code as I am reusing the exact same functions in two different places, making my game more inefficient and may affect performance negatively.

Moving Entity to Standing Player Collision

```

checkCollision(x, y, direction){
    //...
    // Check if new x and y values are within the bounds of the walls
    if (x >= 0 && x < this.walls.width && y >= 0
        && y < this.walls.height) {
        // Check for collision with entities, except the player
        Object.values(this.entities).forEach(entity => {
            if(x == entity.x/16 && y == entity.y/16){
                collide = true
            }
            //If there is a player in front then collide
            if(x == player.x/16 && y == player.y/16){
                collide = true
            }
        })
    };
    // Return the final value of collision
    return collide;
}
//...

```

Map.js

I have updated the checkCollision() method to allow the entity to collide with the player in the game; if the entity is about to travel to a tile where the player object is currently stationed, the entity's movement will be prevented, creating a collision from entity to player. This allows the NPC or monster entity to collide with the player as well as map objects only using one function.

I could use a different alternative approach by using a conditional statement to determine whether the entity being checked for collision is the player entity, which if is the case, then the loop iteration will be skipped and the game will move onto checking the collision of another entity. This approach will indeed remove the unnecessary executions of code as the game will skip to the next entity straight away if the condition is met, however, my original approach is more efficient and simpler because it only checks for collisions with the player entity rather than having to execute multiple conditional statements which could reduce code efficiency.

Moving Entity to moving entity collision

```

// Check for collision with entities, except the player
Object.values(this.entities).forEach(entity => {
    //Player to entity collision
    if(x == entity.x/16 && y == entity.y/16){
        collide = true }
    //Entity to player collision
    if(x == player.x/16 && y == player.y/16){
        collide = true }

    // Check if the object is the player and there are tiles left
    if (obj.isPlaying && entity.TilesLeft > 0) {
        // Check if the object collides with an entity in the specified direction
        if (obj.y/16 <= (entity.y/16) - yValue && y >= (entity.y/16) + yValue*2 &&
            obj.x/16 <= (entity.x/16) - xValue && x >= (entity.x/16) + xValue) {
            collide = true;
        }
    }

    // Check if the object is not the player
    if (!obj.isPlaying) {
        // Check if the object collides with the player in the specified direction
        if (obj.y/16 >= (player.y/16) - yValue && y <= (player.y/16) + yValue*2 &&
            obj.x/16 >= (player.x/16) - xValue && x <= (player.x/16) + xValue) {
            collide = true;
        }
    }

    // Return the final value of collision
    return collide;
}

```

I have implemented a collision system for moving entities colliding with a moving entity, which could also include NPCs, monsters and the player. When iterating through each entity, if the object that is passed in to the function as a parameter is the player instance and the entity instance that is currently being selected from the **this.entities** using the **forEach()** function, and its checking if that entity has tiles left to travel. If that is the case, the method is going to check if the moving entity's next coordinate bounds interfere with the player's next coordinate, then the object that is closer to the next tile is able to proceed, whereas the other colliding entity will stop. I have implemented the same logic outside of the **forEach()** function where it checks the collision between two moving objects, however, this time it will stop the entity from travelling meaning they will collide instead of the player.

Test

Test No.	What I am testing	Test code	Result Required
1	Check if the NPC is moving with the correct conditions and collides with the map walls.	Obj.update() checkCollision()	NPC collides with walls when moving in the direction of the wall and doesn't walk over or through it
2	Check if the NPC collides with any object when is initialised	Obj.update() checkCollision()	NPC collides with any any nearby objects during initialisation and doesn't walk over or through it
3	Check if a moving NPC collides with the standing player	Obj.update() checkCollision()	NPC collides with standing player and doesn't walk over or through it
4	Check if a moving NPC collides with a moving player	Obj.update() checkCollision()	Moving NPC and a moving player collide with each other allowing only one of them to proceed with their walking behaviour while the other stops.

The first test is an **erroneous** test that determines whether the NPC collides with the map walls, when the tiles left to travel are greater than the distance between the NPC and the nearest map wall. It tests the functionality of the **checkCollision()** method and its effectiveness by verifying this method is able to deduce when the NPC is colliding with the map wall and verifies if this method is able to prevent the NPC from moving further to the colliding direction. This test is important because it ensures that the NPC's movement behaviour is robust and compatible with the collision feature of this game.

The second test is a **normal** test that determines whether the NPC still collides with any map walls when it has tiles left to travel to a direction where there is a map wall and the NPC is deployed right in front of the map wall initially. I could skip this test, however, this test will ensure and provide information about the robustness of this method, and whether there are any errors in different scenarios.

The third test is also a **normal** test that determines whether a moving NPC will collide with a standing player, which is important as it tests the functionality that I have developed in the **checkCollision()** method.

The fourth test is a **boundary** test because it simulates the collision of a moving NPC with a moving player. This test is really important to ensure that the **checkCollision()** method is functional and provides collision for this complex scenario.

1st Test: First Iteration

For this test, I have set TilesLeft to 20, meaning that '**npc1**', from the **StartingTown** map, will be required to travel 20 tiles to the default direction which is '**down**'. I have updated the code in the Global.js as seen in the code snippet below;

```
windowMaps{
  //...
  StartingTown: {
    name: "StartingTown",
    entities: {
      npc1: new Obj({
        //sets NPC1 properties
        name: "Boy",
        x: 25, y: 16,
        TilesLeft: 20,
        Direction: 'left',
      }),
    //...
  }
}
```

This should be enough to allow the NPC to carry out their movement according to their attributes set in the **Globals.js** file. This test is erroneous because I am testing with invalid numbers, as the NPC will not be able to travel 20 tiles because I am expecting it to stop after about 8 tiles as there should be a collision with the nearest map wall. Therefore, this test is checking how the game handles with invalid data assigned to the **this.TilesLeft** attribute.

1st Test: First Iteration Test Results



The test was unsuccessful as the entity NPC that I am testing is not travelling, therefore the test was unsuccessful as I cannot test if the entity is able to travel when their TilesLeft attribute is greater than 0.

Error

After some careful debugging, I have concluded that the error was that the entity positions were being updated before the game even checked for collisions. As a result, it caused some issues with the collision handling.

1st Test: Second Iteration

```
//A blueprint for an object in the game
class Obj {
    //...
    //The Update method will commit changes to the object
    update(state) {

        //If the entity has tiles left to travel...
        if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16)
            && this.tilesLeft > 0){
            //If there is a collision tile in front of the player
            if(state.map.checkCollision(this.x/16, this.y/16, this.direction)){
                //Stop the entity by updating the TilesLeft and behaviour
                this.tilesLeft = 0;
                this.behaviour = "standing";
                //Otherwise, let the entity keep travelling
            } else{
                this.tilesLeft = this.speed * 16;
                this.behaviour = "walking";
            }
        }
        //Call the entity movement methods
        this.updatePos()
        this.updateSprite()
    }
    //...
}
```

Entity.js

I have rearranged the order of the code in the update() method, by placing the entity movement methods after checking the collision, which should resolve my initial error.

1st Test: Second Iteration Test results



The test was unsuccessful as I have the same result of the NPC not moving again.

However, the reasons for the rearrangement of the calling of the two methods will still be left like that because it is more suitable for the game to update the entity's position before updating their sprite, which will allow the entities to still update their sprite when it is colliding, providing an effect where the entity is trying to move in the direction that it is colliding but it is being constantly pushed back.

1st Test: Third Iteration

```
checkCollision(x, y, direction){
    // Set initial value of collision as false
    let collide = false;

    //...
    if (x >= 0 && x < this.walls.width && y >= 0 && y < this.walls.height) {
        // Check if there are any walls defined
        if (Object.keys(this.walls).length !== 0) {
            // Check if the current position contains a wall
            if (this.walls.data[x + (y * this.walls.width)] !== 0) {
                //If so, then let the entity to collide
                collide = true;
            }
        }
        //...
    }
}
```

I may have the solution to this issue, which was to default the **collide** variable to false, meaning that the player can always travel to positions unless there is a map wall or entity object which causes a collision. This may resolve the problem because currently, the NPC is unable to move even though there is no collision in any tile adjacent to their position.

1st Test: Third Iteration Test Results



The test was unsuccessful as I still have the same issue of the NPC not moving.

However, It is better to keep this minor change of defaulting the **collide** variable to false, rather than defaulting the entities to not move for every instance, which will lead to less errors in further development.

1st Test: Fourth Iteration

```
//A blueprint for an object in the game
class Obj {
    //...
    //The Update method will commit changes to the object
    update(state) {
        //If the entity has tiles left to travel...
        if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16)
            && this.TilesLeft > 0){
            //If there is a collision tile in front of the player
            if(state.map.checkCollision(this.x/16, this.y/16, this.direction)){
                //Stop the entity by updating the TilesLeft and behaviour
                this.TilesLeft = 0;
                this.behaviour = "standing"
                //Otherwise, let the entity keep travelling
            } else{
                this.TilesLeft = this.speed * 16;
                this.behaviour = "walking";
            }
        }
        //Call the entity movement methods
        this.updatePos()
        this.updateSprite()
    }
    //...
}
```

Entity.js

I believe that the `checkCollision` method works properly because of the previous test I have conducted before using the player instance, which leads me to believe that the issue remains within the `Obj.update()` method itself instead. The 'else' statement in the code snippet above is preventing the NPC from moving anywhere in the map because this statement was preventing the entity from updating its movement when there was no collision detected.

1st Test: Fourth Iteration



[1st Test: Fourth Iteration Test Video](#)

As seen in the video, I have tested the same code three times to check for any possible errors etc. The NPC keeps moving until it reaches a map wall even though it has tiles left to travel as the `TilesLeft` attribute is set to 20, but the NPC is travelling only 5 tiles because a map object has interfered with its movement.

The test was successful.

2nd Test: First Iteration

In this test, I will be testing if the NPC collides with a wall when it has tiles left to travel and is initialised in front of the wall. This test will ensure the robustness of this feature. I have added the changes in the following code snippet temporarily for this test.

```
window.Maps{
    //...
    StartingTown: {
        name: "StartingTown",
        entities: {
            npc1: new Obj({
                //sets NPC1 properties
                name: "Boy",
                x: 23, y: 16,
                TilesLeft: 20,
                Direction: 'left',
            }),
            //...
        }
    }
}
```

Globals.js

I have configured the position of **npc1** to be in front of a map wall, meaning that this entity should collide as soon as the game is initialised.

2nd Test: First Iteration Test Results

2nd Test: First Iteration Video



The test was unsuccessful, the NPC could walk right through the wall when it is initialised moving towards a map wall.

Solution

The way to resolve this is to have the game initialising the `fetchCollision()` and making sure the code inside of this method has been executed before method before initialising the game loop

2nd Test: Second Iteration

As I mentioned before, I have made sure that the current map's collisions are fetched and initialised before initialising the other classes and functions, as seen in the following code snippet, changes highlighted in green:

```
class Game{
    //...
    //The init method will Initiate the game
    init() {
        //Initialises the starting map
        this.map = mapDict.StartingTown;
        //Gets map coordinates to check for collisions
        this.map.fetchCoordinates().then(() => {
            //Initialises a keyInput instance
            this.keyInput = new keyInput();
            this.keyInput.init();
            //Initialises the game Loop
            this.Loop();
        });
    }
}
```

Game.js

I have made a change to the **fetchCoordinates()** method call in the **Game.init()** method, where it returns the coordinates of all collision tiles on the current map as a Promise.

When the **fetchCoordinates()** method is called and resolved, meaning that all the coordinates for the collision have been fetched, then the **init()** method will then proceed on initialising other classes such as the keyInput instance and starting the **Game.Loop()**. This solves the issue because the **update()** methods for NPCs are in the **Game.Loop()**, meaning that the collision coordinates are fetched and resolved before the NPCs are updated or initialises their movement features etc.

2nd Test: Second Iteration Test Results



The test was successful, the NPC now collides with a map wall when initialised and programmed to move towards it, meaning there is no movement from the entity that is being tested.

I can now proceed to the next test.

3rd Test: First Iteration

The third test will ensure that the entity collides with the player when the player immobile. To carry out the test, I have updated the attributes of the npc1 from the StartingTown map where it will collide with the player eventually as it travels down the map.

Changes in green in the following code snippet:

```
//Creates global Maps object
windowMaps = {
  StartingTown: {
    name: "StartingTown",
    entities: { //Collection of entities of StartingHouse map
      npc1: new Obj({
        //sets NPC1 properties
        name: "Boy",
        x: 25, y: 10,
        TilesLeft: 20,
        direction: 'down'
      }),
    }
  }
  //...
}
```

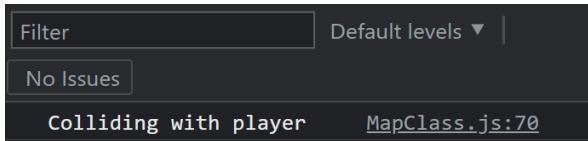
Global.js

To check if the entity is colliding with the player, I have added a console.log function in the checkCollision() method, which will be executed if the player is on the tile that the entity was going to travel to. Here's the following temporary addition for this test:

```
checkCollision(x, y, direction){
  //...
  // Check for collision with entities, except the player
  Object.values(this.entities).forEach(entity => {
    if(x == entity.x/16 && y == entity.y/16){
      collide = true
    }
    if(x == player.x/16 && y == player.y/16){
      console.log("Colliding with player")
      collide = true
    }
  })
  //...
}
```

Map.js

This console log function determines whether the NPC is colliding with the player..

3rd Test: First Iteration Test Results[3rd Test: First Iteration Test video](#)

The test was successful as seen in the video.

The NPC successfully collides with the player and the console prints out 'Colliding with player' when this action occurs.

Refactoring

```
class Map(){
    //...
    checkCollision(obj){
        // Destructure x and y values from the dictionary based on the input
        const [xValue, yValue] = directionDict[obj.direction];

        // Update x and y values based on direction
        const x = obj.x/16 + xValue;
        const y = obj.y/16 + yValue;

        //...
        if(player.TilesLeft > 0){
            const [axis, value] = player.directionDict[player.direction];
            console.log(axis)
        }
        //...
    }
    //...
}
```

I have refactored the following code by passing in an instance rather than separate parameters and variables, which reduces the length of the code and increases efficiency of the code.

Moreover, passing on the instance will also give me access to more variables rather than passing on the required parameters. I have got the x and y value of the instance by destructing using the directionDict and the object's direction.

Then I have created two new variables which are the coordinates of where the instance is going to travel, as a result, I will now not be required to pass in any additional variables.

```
class Obj {
//...
//The Update method will commit changes to the object
update(state) {
    //If the entity has tiles left to travel and collides...
    if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16) &&
        state.map.checkCollision(this)){
        //Prevent the entity from travelling
        this.TileLeft = 0;
    }
    //Update entity position and sprite
    this.updatePos()
    this.updateSprite()
}
}
```

```
class Player{
//...
update(state){
    //Updates player's direction when TileLeft is 0
    if (this.TileLeft === 0 && state.direction) {
        this.direction = state.direction;
        if (state.map.checkCollision(this)) {
            this.TileLeft = 0;
            this.behaviour = "standing"
        } else {
            this.TileLeft = this.speed * 16;
            this.behaviour = "walking";
        }
    }
    //...
}
}
```

Because of the previous refactoring, I will need to only pass in the instance that it requires. In this case, I can just pass in the keyword "this". This makes my game more efficient and allows other functions from different classes to access more information about the entity such as its attributes.

I could have not passed an entity at all as a parameter, and instead made each entity object a global object such as the player instance to simplify the code even more.

However, this approach is not maintainable and will create more unnecessary global dependencies in the code which could severely affect the performance of the game negatively.

4th Test: First Iteration

```

checkCollision(obj){
    //...
    // Check for collision from player to entity
    Object.values(this.entities).forEach(entity => {
        //If the object is the player and it has tiles left to travel
        if(obj.isPlayer && entity.TilesLeft > 0){
            //If the player is within the bounds of where an entity has already
            travelled to
            if(obj.y/16 <= (entity.y/16) - yValue && y >= (entity.y/16) + yValue*2
                &&
                obj.x/16 <= (entity.x/16) - xValue && x >= (entity.x/16) + xValue){
                    //Then stop the player because it has intersected the tile latest
                    collide = true;
                    console.log("Player colliding with NPC!")
                }
            }
        })
        //Check collision from the entity to player
        if(!obj.isPlayer){
            //If the entity is within the bounds of where the player has already
            travelled to
            if(obj.y/16 >= (player.y/16) - yValue && y <= (player.y/16) + yValue*2
                &&
                obj.x/16 >= (player.x/16) - xValue && x <= (player.x/16) + xValue){
                    //Then stop the entity because it has intersected the tile latest
                    collide =true;
                    console.log("NPC colliding with player!")
                }
            }
        }
    //...
}

```

The conditional statements' console log statements are used for the purposes of this temporary test. The game will log "Player colliding with NPC!" to the console if the first if statement's condition is satisfied, indicating that a player object and an NPC object are colliding. And the game will also log "NPC colliding with player!" to the console, indicating that an NPC object is colliding with the player object, if the condition inside the second if statement is satisfied. These console logs can be used to understand the way the algorithm works and spot any collision detection problems. The log statements may be deleted once the code has been verified to be accurate. For the test to be completely successful, the game will need to log both statements simultaneously and each statement must be only logged once.

4th Test: First Iteration Test Results[4th Test: First Iteration Test Video](#)

Test No.	Console results	Video Results
1.	<pre>NPC colliding with MapClass.js:98 player!</pre>	
2.	<pre>Player colliding with MapClass.js:88 NPC! NPC colliding with MapClass.js:98 player!</pre>	
3.	<pre>6 Player colliding with MapClass.js:88 NPC! NPC colliding with MapClass.js:98 player!</pre>	

I have received inconsistent results throughout this test, however I did get the correct console results that I was looking for which was the first try in the table above.

However, the issue was that the NPC will stop for a moment but the player will keep moving on instead of both NPC and player stopping.

Therefore, **the test is partially successful**, but I will need to complete it successfully in order to meet the success criteria

4th Test: Second Iteration

```

// Check if the given object collides with any entities
checkCollision(obj){

    // Calculate the new x and y values based on the object's direction
    const x = Math.round(obj.x/16 + xValue);
    const y = Math.round(obj.y/16 + yValue);

    //...

    // Check for collision with entities, except for the player
    Object.values(this.entities).forEach(entity => {
        // Check for collision between the player and the entity
        if(x == Math.round(entity.x/16) && y == Math.round(entity.y/16)){
            collide = true
        }
        // Check for collision between the entity and the player
        if(x == Math.round(player.x/16) && y == Math.round(player.y/16)){
            collide = true
        }

        // Check for collision between the player and a moving entity
        if(obj.isPlayer && entity.behaviour === "walking"){
            // Calculate the new x and y values for the entity
            const [entityXValue, entityYValue] = directionDict[entity.direction];
            const newEntityX = entity.x/16 + entityXValue;
            const newEntityY = entity.y/16 + entityYValue;

            // Check if the player's x-coord is between the entity's current and
            new x-coord
            if((x >= newEntityX && x <= entity.x/16) ||
                (x >= entity.x/16 && x <= newEntityX)){
                // Check if the player's y-coord is between the entity's current and
                new y-coord
                if((y >= newEntityY && y <= entity.y/16) ||
                    (y >= entity.y/16 && y <= newEntityY)){
                        collide = true;
                    }
                }
            }
        })
    }
}

```

```

// Check for collision between a non-player object and a moving player
if(!obj.isPlayer && player.behaviour === "walking"){
    // Calculate the new x and y values for the player
    const newX = Math.round(player.x/16 +
directionDict[player.direction][0])
    const newY = Math.round(player.y/16 +
directionDict[player.direction][1])
    // Check if the object's x and y values match the player's new x and y
values
    if( x == newX && y == newY){
        collide = true;
    }
}

// Return true if a collision was detected, false otherwise
return collide;
}

```

I have adjusted to solve some of the difficulties that existed in the previous implementation of the **checkCollision()** method which was leading to some inconsistency and errors during the testing process. When the x and y coordinates were assigned to the current entity object, a **Math.Round()** function has been used to round the coordinates of the entity to the nearest integer, which avoids the problem when checking for collisions for a wall or entity. The code then checks for collisions between entities and the player, again using the **Math.Round()** function. As both the player and the entity are moving, the function checks for collisions, by examining that entity's and player's direction and position and computing their new position based on their direction of movement. If one entity's new location overlaps or collides with the position of the entity, the entity which came 'late' to the tile is going to be stopped while the other entity will proceed to move as normal.

The new code should fix the inconsistency as I have figured out that the previous **checkCollision()** function version wasn't working properly because the collision was not being checked at the right time and at the right conditions.

One of the enhancements and improvements I have made in the new **checkCollision()** method is that I have used **Math.round()**, which is being used to round the coordinates of the entities to the nearest integer. This ensures that the collisions are correctly detected at the right conditions and at the right time, as it does not skip over any collisions between the moving entities because of floating-point imprecision.

Another improvement that I have implemented into the new function is that the game checks the entity's next whole tile, whereas in the previous version, the next tile that one of the entity were going to travel was not an integer number, meaning that the coordinates of the new tile that one of the entity is about to travel to changes with every iteration and the coordinates are

represented as floats rather than whole numbers or integers. This again causes floating point imprecision.

Finally, I have simplified and refactored some elements of the code by removing unnecessary conditional statements, such as verifying if the wall object is empty and using destructuring.

Overall, the new `checkCollision()` method ensures that the moving collision feature is robust and works perfectly.

I could have used a simpler algorithm that takes fewer computer resources that could be used as an alternative approach to the new codes of the `checkCollision()` method.

Employing a simpler approach can help to conserve computational resources which is useful to improve the efficiency of the game and may also enhance performance as the code will execute fewer functions which will save more time.

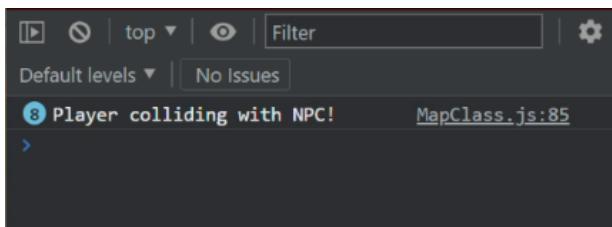
However, this alternative approach may not always be accurate because simpler algorithms may not be capable of handling increasingly complex issues like mine.

My improved method is more complex and requires more computational resources, but it is optimised for the specific situation which is the collision between two moving entities.

This method is more precise and effective than the alternative and the previous version of the `checkCollision()` method, as this newer approach is far more superior because it is more accurate and vital in instances where the outcome is critical. Furthermore, the new approach might aid the moving entities collision situation, even though it is using more computational resources than like a simpler algorithm, the benefits of the new approach outweigh the drawbacks of the new method and benefits of the previous and alternative approaches.

4th Test: Second Iteration Test results

[4th Test: Second Iteration Test Video](#)



A screenshot of a terminal window. At the top, there are icons for play, stop, and filter, followed by the text "Default levels ▾" and "No Issues". Below this, the message "8 Player colliding with NPC! MapClass.js:85" is displayed in green text. To the right of the terminal window, the text "The test was successful" is written in green.

The test was successful

As seen in the video, when the player and NPC moves towards the same tile, the entity which moves to that tile first will be able to proceed, whereas the entity will be stopped and marked as colliding which is shown in the console log.



Refactoring

When an NPC or monster collides with a moving entity, I want them to resume their movement as they were moving before when it stops colliding with the moving entity, meaning that the moving entity is not on their way anymore.

```
class Obj{
    constructor(config) {
        if (this.TilesLeft > 0){
            this.behaviour = "walking"
        } else {
            this.behaviour = "standing";
        }
        //...
    }
    //...
    //The Update method will commit changes to the object
    update(state) {
        //If the entity is standing on a whole tile and needs to move
        if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16) &&
            this.TilesLeft > 0 ){
            //Check if they are colliding...
            if(state.map.checkCollision(this)){
                //If so, switch the behaviour to standing to stop the entity
                //from moving
                this.behaviour = "standing"
            }
        }

        if(this.TilesLeft === 0){
            this.behaviour = "standing"
        }

        this.updatePos()
        this.updateSprite()
        //this.startBehaviourLoop()
    }
    //...
}
```

I have used the **this.behaviour** attribute to determine whether an entity is capable of moving depending on the result from calling the **checkCollision()** method. I could have kept the other approach of using **this.TilesLeft** to determine whether the player can move or not after a collision detection has been verified, as this approach keeps the code more simple and does not require any new attributes, however, Instead of using **this.TilesLeft** to check if the entity is moving or not and to assign if the entity is allowed to move not, I have used **this.behaviour**.

This alternative approach is far more efficient because it eliminates the need for unnecessary calculations and comparisons required when using the **this.TileLeft** attribute.

This.behaviour is directly assigned in the specified code based on the value of **this.TileLeft** during the object's initialisation in the constructor. Therefore, **this.behaviour** is determined once at the entity initialisation and the code does not need to verify the **this.behaviour** attribute several times in the **update()** method which potentially leads to a faster and more efficient code execution. Furthermore, when compared a numeric attribute such as **this.TileLeft**, the **this.behaviour** attribute is more legible and easier to understand in the perspective of other developers as it stores a more descriptive property within itself. While deciding between different alternatives, it is essential to evaluate the overall design and performance of implications. Generally, **this.behaviour** is being used as a replacement for **this.TileLeft** only when it comes to determining whether the entity instance can move or not.

```
updatePos() {
    if (this.behaviour === "walking") { //If the player has to move
        //Checks which direction it needs to move to
        const [axis, value] = this.directionDict[this.direction]
        //Changes their position value on the correct axis
        this[axis] += value * this.speed
        // If this method continues to run, the method will stop when the
        //TilesLeft is 0
        this.TileLeft -= this.speed * this.speed;
    }
}
```

As seen in the code snippet above regarding the **updatePos()** method, the method is determining whether the player should currently be walking or standing. If **this.behaviour** is 'walking', then proceed with the rest of the calculator on updating the player's position, otherwise, none of the processes of the updatePos() will be executed.

Furthermore, this change is only applied to the Obj class, the player class is not affected by this change, therefore, the player class will now require a different updatePos() method and cannot inherit from the entity class. As a result, the player class and the entity class will have **different methods and the inheritance of the updatePos() is broken**.

Behaviour Loops

A Behaviour loop is a sequence of instructions that establishes an NPC's behaviour throughout the game. It includes a number of behaviours or actions, like walking or standing stationary, as well as a number of specifications, including the direction, speed, and duration. These actions are carried out in a loop progressively, giving the NPC the appearance of constant movement or activity. The behaviour loop makes it possible for players to have a more dynamic and realistic gaming experience since NPCs may move and interact with their surroundings in a natural way.

Handling walking behaviours

```
class Obj{
    constructor(config) {
        //...
        this.behaviourLoop = config.behaviourLoop
        this.currentBehaviour = -1;
        this.time = 0

        //The Update method will commit changes to the object
        update(state) {
            if(this.tilesLeft > 0){
                this.behaviour = "walking"
            }
            if(Number.isInteger(this.x/16) && Number.isInteger(this.y/16) &&
this.tilesLeft > 0 ){
                if(state.map.checkCollision(this)){
                    this.behaviour = "standing"
                }
            }
            if (this.time > 0) {
                setTimeout(() => {
                    this.time = 0;
                }, this.time);
            }
            this.updatePos()
            this.updateSprite()

            if(this.tilesLeft === 0 && this.time === 0){
                this.currentBehaviour++;
                this.startBehaviourLoop()
            }
        }
        //...
    }
}
```

Entity.js

I have implemented this feature in the Obj class, firstly by initialising a few attributes such as behaviorLoop attribute, which is an array that will store a set of instructions that define the NPC's behaviour over time. Secondly, the currentBehaviour will keep track of the behaviour that the entity is executing at the current moment, this attribute is stored as an index. Thirdly, the time attribute is initialised to handle the standing behaviours in the behaviour loop feature, which is used to delay the execution of the standing behaviour for a specific time, this attribute will store an integer as milliseconds.

I have added changes to the update() method of the Obj class, firstly by assigning the correct behaviour to the entity by checking for their **this.tilesLeft** attribute using a conditional statement, if **this.tilesLeft** > 0 then set the behaviour attribute to "walking", if **this.tilesLeft** < 0 or the entity is colliding, then set the behaviour attribute to "standing".

I have used the setTimeout() function to delay execution and handle the standing behaviours using **this.time** which specifies how long the behaviour should be delayed by.

The third change that I have added to the update() method is checking if **this.tilesLeft** and **this.time** equals to 0, if that is the situation then the game will move on to the next behaviour as structured in the **this.behaviourLoop** dictionary, indicating that the current behaviour that was being executed has finished, whether it was a standing behaviour or walking behaviour. After that a **startBehaviourLoop()** method is called...

```
startBehaviourLoop() {
    // Check if there is a behaviour loop
    if (this.behaviourLoop) {
        // Reset the current behaviour if it has exceeded the behaviour Loop Length
        this.currentBehaviour = this.currentBehaviour % this.behaviourLoop.length

        // Set the behaviour and direction
        const { behaviour, direction, time, tiles } =
            this.behaviourLoop[this.currentBehaviour]
        this.behaviour = behaviour
        this.direction = direction

        // Update TilesLeft and time based on the behaviour
        if (behaviour === "standing") {
            this.time = time
            this.tilesLeft = 0
        } else {
            this.tilesLeft = tiles * 16
            this.time = 0
        }
    }
}
```

Entity.js

The code snippet demonstrates a startBehaviourLoop() method that starts the subsequent behaviour of an Obj class object's behaviour loop. The method initially determines if the object has a defined behaviour loop. If a behaviour loop exists, it resets the currentBehaviour property if it has exceeded the behaviour loop length. Using the behaviourLoop array, the procedure then sets the behaviour and direction attributes. The TilesLeft property is set to 0 and the time property is set to the time value of the current behaviour if the behaviour is "standing." The time property is set to 0 and the TilesLeft property is set based on the current activity's tile value if the behaviour is "walking."

```
entities: {
    npc1: new Obj({
        behaviourLoop: [
            {behaviour: "walking", direction: "down", tiles: 5},
            {behaviour: "standing", direction: "left", time: 5000},
            {behaviour: "walking", direction: "right", tiles: 5},
            {behaviour: "standing", direction: "up", tiles: 3000},
        ],
    }),
}
```

Example of configuring behaviourLoop

Here's an example of the behaviourLoop attribute where this array attribute of the npc1 object lists the behaviours that should be displayed by it sequentially. The npc1 object in this instance will move five tiles down, five tiles to the left, five tiles to the right, five tiles up, and three tiles up before standing up for three seconds. It will repeat the cycle from the beginning once it has finished performing all of these activities. This makes it possible to create a wide range of alternative behaviours for an object in a 2D RPG game.

Test

Test No.	What I am testing	Test Data	Result Required
1	The NPC should be able to walk up and down	StartBehaviourLoop() Entity.update() Entity.behaviourLoop this.tilesLeft	The NPC should repeatedly travel back and forth between two points, travelling a specific amount of tiles in each direction. The this.tilesLeft property keeps track of the remaining distance, and when it reaches 0, the NPC should move to the next behaviour. If the NPC completes the last behaviour in the behaviourLoop dictionary, the game should restart the loop from the beginning.
2	The NPC should be able to switch directions while standing	StartBehaviourLoop() Entity.update() Entity.behaviourLoop this.time	When still standing, the NPC should change its direction before moving on to the following action in the loop.
3	The NPC should be able to walk and stand in a repetitive sequence	StartBehaviourLoop() Obj.update() Obj.behaviourLoop this.time this.tilesLeft	The NPC should continue performing the prescribed set of walking and standing behaviours indefinitely or until anything external interrupts it.

The test plan will aim to iteratively test each component of the behaviour loop feature. The NPC must repeatedly go back and forth between two places while moving a certain number of tiles in each direction. Before turning around, we need to make sure the NPC moves smoothly and stops at the appropriate locations. Also, I need to ensure that the NPC pauses and switches course when it reaches the end of its path and advances the appropriate number of tiles in each direction. Moreover, We must confirm that the NPC can change directions while standing. I need to make sure the NPC can change directions without any problems and that it stops going when it reaches the end of its path. Finally, I need to ensure that the NPC can walk and stand in a repetitive sequence. I must ensure that the NPC moves and stops at the correct points along its path and that the sequence is repeated without error. We should also ensure that the NPC's sprite is correctly updated to reflect its movement and standing states.

The test code in these tests are mostly the same as the **Obj.behaviorLoop** attribute will be configured accordingly to what I am testing, this attribute will then be passed into the **update()** method, which will call the execute the delay process of the standing behaviours using the **this.time** attribute and call the **StartBehaviourLoop()**, which will iterate through the **this.behaviourLoop** attribute that contains a set of behaviours to be executed for that entity.

1st Test: First Iteration

```
window.Maps = {
  //...
  StartingTown: {
    name: "StartingTown",
    entities: { //Collection of entities of StartingHouse map
      npc1: new Obj({
        //sets NPC1 properties
        name: "Boy",
        x: 25, y: 10,
        behaviourLoop: [
          {behaviour: "walking", direction: "down", tiles: 9},
          {behaviour: "walking", direction: "up", tiles: 9},
        ]
      }),
    }
    //...
  }
}
```

Global.js

The behaviour of **npc1** in the StartingTown map is defined by an array of objects called the **behaviourLoop** property. The array's objects each represent a distinct behaviour and have the following properties:

- **Behaviour:** A string indicating the kind of behaviour to display.
For the purpose of this test, npc1 is going to "walk" only.
- **Direction:** A string indicating the entity's preferred motion direction.
The direction can be "up," "down," "left," or "right" as usual.
- **Tiles:** An integer that indicates how many tiles the entity should travel in the given direction before coming to a stop and switching direction

The npc1 object entity in the above code snippet has a **behaviourLoop** property that contains two behaviour objects. The NPC is told to move nine tiles down in the first object and nine tiles up in the second. As a result, the NPC behaves in a looping manner by moving up and down endlessly. This type of behaviour loop can be used to produce simple, repetitive activities for game entities, such as strolling back and forth or patrolling a space. Instead of specifying the number of tiles the npc should go, I could use another strategy for this feature, such as adding the coordinates for the destination instead. This alternative strategy offers more flexibility and enables pathfinding, which could be quite useful in this scenario. But, for a straightforward 2D single-player game like mine, my original strategy is better because it is much simpler to implement and comprehend.

```
//The Update method will commit changes to the object
update(state) {
    //...
    console.log("TilesLeft: "+Math.round(this.TilesLeft/16),
        "\nindex:",this.currentBehaviour,
        "\nBehaviour:",this.beaviour)
    //...
}
```

Entity.js

The console log in the update method of the object is printing the value of three properties.

The first property, TilesLeft (where I have used a Math.round() function) is split by 16 and rounded to the nearest integer to simplify things.

CurrentBehaviour, which represents the index of the current behaviour being carried out by the object, is the second property.

The name of the behaviour that the object is carrying out is represented by its third property, behaviour. Debugging and comprehending the behaviour of the object during runtime are both possible with the help of this console log.

I am expecting the behaviourLoop feature to work correctly and expecting the result to meet the following criterias:

1. The npc must travel back and forth repeatedly
2. The npc must travel 9 tiles in each direction
3. When this.TilesLeft decreases to 0, the entity should move on to the next behaviour by increasing the this.currentBehaviour index.
4. If the npc executes the last behaviour in the behaviourLoop attribute, then it must restart the loop from the first behaviour, setting this.currentBehaviour index back to 0.

1st Test: First Iteration Test Results

16	TilesLeft: 2 index: 0 Behaviour: walking	Entity.js:156
16	TilesLeft: 1 index: 0 Behaviour: walking	Entity.js:156
7	TilesLeft: 0 index: 0 Behaviour: walking Running	Entity.js:156 Entity.js:170
9	TilesLeft: 9 index: 1 Behaviour: walking	Entity.js:156
8	TilesLeft: 8 index: 1 Behaviour: walking	Entity.js:156

[1st Test: First Iteration Test Video](#)

The test was successful.

As seen in the video and in the console log, I have met all the criterias for this test. The npc travels back and forth for 9 tiles in both directions while their this.TilesLeft decreases until 0, which then proceeds to the next behaviour and increases the index, which represents the behaviour count.

2nd Test: First Iteration

```
window.Maps = {
  //...
  StartingTown: {
    name: "StartingTown",
    entities: { //Collection of entities of StartingHouse map
      npc1: new Obj({
        //sets NPC1 properties
        name: "Boy",
        x: 25, y: 10,
        behaviourLoop: [
          {behaviour: "standing", direction: "down", time: 1000},
          {behaviour: "standing", direction: "right", time: 1000},
          {behaviour: "standing", direction: "left", time: 1000},
          {behaviour: "standing", direction: "up", time: 1000},
        ]
      }),
    }
    //...
  }
}
```

Globals.js

The code snippet's behaviourLoop attribute has been changed for the purposes of this test, whereas now it specifies a looping set of behaviours for an NPC, more specifically, a series of directions the NPC will face while remaining still for a predetermined period of time. The NPC will begin by facing downward and remain stationary for one millisecond (Or 1000 millisecond) before turning to the right and being motionless for one more second again. This process will continue until all directions have been cycled through, at which point the sequence will restart. This looped behaviour can produce an NPC movement pattern that is predictable and repetitious, which can enhance the atmosphere of the game world.

```
//The Update method will commit changes to the object
update(state) {
  //...
  console.log("Time:",this.time,
  "\nIndex:",this.currentBehaviour,
  "\nBehaviour:",this.beaviour)
  //...
}
```

Entity.js

Similarly to the previous test, I will be required also to use a console.log() function for debugging and testing to see if this second test meets the criteria for success.

2nd Test: First Iteration Test Results

```
Time: 1000          Entity.js:160
index: 0
Behaviour: standing

Time: 1000          Entity.js:160
index: 1
Behaviour: standing

Time: 1000          Entity.js:160
index: 2
Behaviour: standing

② Time: 1000        Entity.js:160
index: 3
Behaviour: standing

② Time: 1000        Entity.js:160
index: 0
Behaviour: standing
```

2nd Test: First Iteration Test Video

The test was unsuccessful, as seen in the video the npc will iterate through the behaviour loop really quickly and less than 1 second meaning that the expected results did not correspond with the actual results. Moreover, in the console, it can be seen that each behaviours have inconsistent length as some behaviours are executed for 'once count' while other behaviours are executed for 'two counts'.

Issue

```
if (this.time > 0)
  setTimeout(() =>
    this.time = 0
  ), this.time);
}
```

I believe that the main cause of this issue is because of the setTimeout() function that I have used. This leads to error because this function is not compatible with the game loop that I have implemented for this game

Solution

```
if (this.time > 0) {
  this.time -= 60
}
this.time = Math.max(this.time, 0)
```

The new approach enhances the previous code by simulating time passing by using a decrementing counter (this.time -= 60), where the value of 60 represents the number of milliseconds since the last loop iteration.

No matter how long the loop iteration takes to finish, this makes sure that the code is run at regular intervals. This.time will never be negative because of the Math.max(this.time, 0) instruction.

I could use a different approach by using the Date.now() method to determine the amount of time that has passed between the current and prior loop iterations as an alternative approach to time-based behaviour in game loops. It offers precise timing and is unaffected by the number of loop iterations. However, this method needs more code than my new solution, and the newer approach is much simpler to understand and only requires three lines of code instead of having to write a whole new function for the alternative approach.

2nd Test: Second Iteration

```
//The Update method will commit changes to the object
update(state) {
    //...
    if (this.time > 0) {
        this.time -= 60
    }
    this.time = Math.max(this.time, 0)
    //...
}
```

Entity.js

I have now implemented the new solution to the update method of the Obj class.

2nd Test: Second Iteration Test Results

```
Time: 100
index: 1
Behaviour: standing

Time: 40
index: 1
Behaviour: standing

Time: 1000
index: 2
Behaviour: standing

Time: 940
index: 2
Behaviour: standing
```

[2nd Test: Second Iteration Test Video](#)

The test was successful.

In this iteration the entity changes direction while being stationary for a configurable amount of time, where the configured time decreases by 60 in each game loop iteration and when it reaches 0 or less than 0, the game then allows the entity to move on to the next behaviour by increasing the index of the **this.currentBehaviour** attribute as seen in the console log test.

3rd Test: First Iteration

```
window.Maps = {
  //...
  npc1: new Obj({
    //...
    behaviourLoop: [
      { behaviour: "walking", direction: "down", tiles: 1 },
      { behaviour: "standing", direction: "down", time: 1000 },
      { behaviour: "walking", direction: "down", tiles: 1 },
      { behaviour: "standing", direction: "down", time: 1000 },
      { behaviour: "walking", direction: "down", tiles: 1 },
      { behaviour: "standing", direction: "down", time: 1000 },
      { behaviour: "walking", direction: "down", tiles: 1 },
      { behaviour: "standing", direction: "down", time: 5000 },
      { behaviour: "walking", direction: "up", tiles: 4 },
    ],
  }),
  //...
}
}
```

In this test, I will be combining both standing and walking behaviours and test if this feature works overall, if that is the case, this feature should be fully completed and available to be merged to the main branch of the game and start implementing more behaviours for different entities in different maps.

3rd Test: First Iteration Test result

[3rd Test: First Iteration Test video](#) - The test was successful.

I have also **successfully completed and met the success criteria** from the analysis stage:

Success Criteria	Evidence	Justification
NPC behaviours and collisions	A video of the NPCs moving in specific patterns and colliding with the player and other entities.	The twelfth criteria ensures that the game will have NPCs with unique behaviours and collisions between other entities and the player, to make the game feel more alive. I could use a NPC behaviour template in order to save effort and reduce the length of the code in the developing process, however, this will make every NPCs behave the same which the user may find boring. Including more authenticity for each NPC will provide a more immersive experience

Refactoring

```

//The Update method will commit changes to the object
update(state) {
    // Check if there are tiles left to walk
    if (this.tilesLeft > 0) {
        this.behaviour = "walking"
    }

    // Check if the object is on a tile and there are tiles left to walk
    if (Number.isInteger(this.x / 16) && Number.isInteger(this.y / 16) &&
        this.tilesLeft > 0) {
        // Check for collision
        if (state.map.checkCollision(this)) {
            this.behaviour = "standing"
        }
    }

    // Decrement the time
    if (this.time > 0) {
        this.time -= 60
    }
    this.time = Math.max(this.time, 0)

    // Update the position and sprite
    this.updatePos()
    this.updateSprite()

    // If there are no tiles left to walk and the time is 0, advance to the
    next behaviour
    if (this.tilesLeft === 0 && this.time === 0) {
        this.currentBehaviour++
        this.startBehaviourLoop()
    }
}

startBehaviourLoop() {
    // Check if there is a behaviour Loop
    if (this.behaviourLoop) {
        //Reset the current behaviour if it has exceeds the behaviourLoop Length
        this.currentBehaviour = this.currentBehaviour %
            this.behaviourLoop.length

        // Set the behaviour and direction
        const { behaviour, direction, time, tiles } =
            this.behaviourLoop[this.currentBehaviour]
}

```

```

this.behaviour = behaviour
this.direction = direction

// Update TilesLeft and time based on the behaviour
if (behaviour === "standing") {
    this.time = time
    this.TilesLeft = 0
} else {
    this.TilesLeft = tiles * 16
    this.time = 0
}
}
}

```

I have refactored the behaviourLoop code, its major changes of this feature, where the entity's position and sprite is updated in accordance with the behaviour loops, and I added comments where is applicable and needed.

One of the changes that I have updated In the refactored version of the startBehaviourLoop() is where the current behaviour loop, the object's behaviour and direction attributes are set where the values for these attributes are extracted from the current element of the behaviourLoop array by the code, which then assigns them to the corresponding properties of the object. With the proper behaviour and direction for the current loop iteration, the object is updated in this manner. This refactored code is easier to understand and maintain than the original.

I could have used an alternative approach such as creating a new class and extracting the behaviour object from it. The behaviour class should be given a next method that returns the subsequent behaviour object in the loop. For each item that requires a behaviour loop, make an instance of the behaviour class. To obtain the following behaviour object in the loop, call the behaviour object's next method in the update method of the object.

Based on the behaviour object that next returns, set the object's behaviour properties.

However, my approach of extracting all the attributes from the behaviourLoop attribute by destructuring assignment is better since it enables you to do so in a clearer and more readable manner. It reduces the need for repeating code and improves the maintainability of the code. When manually appointing each property to its relevant variable will lengthen the code, whereas destructuring assignments can result in better ordered and efficient code overall.

Player Interaction

Interaction

```

class Map {
//...
playerInteraction(state){
    //Dictionary to get the direction vector based on player's direction
    let directionDict = {
        "up": [0, -1], "down": [0, 1],
        "right": [1, 0], "left": [-1, 0]
    };

    //Calculating the potential new position of the player
    const newX = player.x/16 +directionDict[player.direction][0]
    const newY = player.y/16 + directionDict[player.direction][1]

    //Check if the new position is on a grid point
    if(Number.isInteger(newX) && Number.isInteger(newY)){
        //Loop through all entities in the map
        Object.values(this.entities).forEach(entity => {
            //Check if entity is not a monster
            if(entity.type !== "monster"){
                //Check if entity is facing the player
                if(entity.x/16 === newX && entity.y/16 === newY){
                    //Check if player wants to interact with entity
                    if(state.enterBool){
                        //If so, Set entity to be interacting and freeze the
                        //player's movement
                        entity.interacting = true;
                        player.freeze = true
                    } else {
                        //Set entity to not be interacting and unfreeze the
                        //player's movement
                        entity.interacting = false;
                        player.freeze = false
                    }
                }
            }
        })
    }
}

```

Map.js

The **playerInteraction()** method in the Map class takes a boolean enterBool parameter indicating whether the player is attempting to interact with an entity. It calculates the player's new position based on their current position and direction using a dictionary called **directionDict**. If there is an entity at the new position that is not a monster, and the enterBool parameter is true, the entity's interacting attribute is set to true and the player's freeze attribute is set to true, preventing them from moving until the interaction is complete. If enterBool is false, the entity's interacting attribute is set to false and the player's freeze attribute is set to false, allowing them to resume movement. This is a typical gameplay concept that is achieved by looking for player-to-map contacts. In order to advance in the game, the player must interact with NPCs.

I could have used an alternative approach such as creating a nested loop to detect the entity that the player will interact with. After the detection process, the game should execute the player interaction feature by freezing both entities and drawing a dialogue onto the canvas, this approach will definitely improve performance especially on to the later stages of the development phase, because I am going to add a lot of NPCs and other entities.

One of the drawbacks, however, is that it may lead to some unintended computational problems because of the several executions of loops in one single iteration, which may cause it to decrease performance rather than increasing the performance of the game.

Furthermore, a nested loop will also decrease the readability of my code and make debugging or testing more difficult to carry out, which could lead to a rise of limitations regarding the maintenance of the code in post-development. As a result, this alternative approach may be slightly more difficult to implement alongside the behaviour loop features because the NPC's coordinates are dynamic and constantly changing.

```

class Player extends Obj {
    constructor(config){
        this.freeze = false;
        //...
    }
    //...
    update(state) { //Updates the character in each
        // If the player has pressed 'enter' key
        if (selectBoolean) {
            // ... Then check if the player is facing an entity that is not a monster
            if (!state.map.playerInteraction(state)) {
                // If not, then set selectBoolean to false
                selectBoolean = false;
            }
        }
        // Call the playerInteraction function with the state parameter
        state.map.playerInteraction(state);

        // If the player is frozen, update the sprite and return
        if (this.freeze) {
            this.sprite.updateSpriteSet("idle-" + this.direction);
            return;
        }
        //...
    }
}

```

Player.js

As seen previously, the **this.freeze** attribute is taken into account as set as false, this attribute will be used as a flag to allow the game to determine whether the player should be able to move or not. If this attribute is true, this would mean that the player instance is currently within a NPC interaction process. In future development, this attribute may also be useful for creating cutscenes, as it is a common approach in many RPG games to freeze the player character instance during a cutscene. Furthermore, all these processes are executed inside the **update()** method, where I have added some new changes regarding the player becoming frozen when it is interacting with a NPC. Initially I check if the user is holding the 'enter' key by using an if statement to see whether the **selectBoolean** attribute is true or not. If the **selectBoolean** is true, then the method will proceed by allowing the player to interact with the player unless the player is not in front of an entity, otherwise there would be no changes, meaning that the game will continue to run as usual. If there is a non-monster entity in front, then the **selectBoolean** will be set to true, calling the **playerInteraction()** method to execute itself appropriately. This method will change some attributes such as `this.freeze`, as shown in the previous code snippet, if this attribute is true, then the player sprite will become idle and the rest of the update method will prevent it from running, which will disable the player to call methods such as `updatePos()` or even toggling on the speed boost while **this.freeze** is set to true.

```
// OBJECT CLASS
class Obj { //A blueprint for an object in the game
  constructor(config) {
    this.interacting = false;
  }

  //The Update method will commit changes to the object
  update(state) {
    const oppositeDirection = {
      "up": "down", "down": "up",
      "left": "right", "right": "left"
    }
    if(this.interacting){
      this.sprite.updateSpriteSet(
        "idle-"+oppositeDirection[player.direction])
      return
    }

    //...
  }
  //...
}
```

Entity.js

I have added a **this.interacting** attribute which is a boolean that indicates whether the current entity instance is interacting with the player or not.

I have added new logic to the update() method such as the oppositeDirection dictionary which stores, as the name suggests, the opposite direction of each of the 4 directions.

This dictionary is then used for the player interaction conditional statement where the update method checks if the entity is currently interacting, if so, the **update()** method will proceed to change the entity's direction to the opposite direction of the player which will allow it to face towards the player and interact, after this process, the code is then returned preventing the **update()** to fully finish the rest of the code, meaning that the entity that is being interacted is now in a freezing state. However, unlike the player class, I do not require an additional attribute for freezing the entity during the interaction process. I could have added a freezing attribute to prevent the entity to be updated any further, however, that would just unnecessary lengthen the code, and this approach is far superior because it is simpler.

```

class keyInput{
    constructor(){
        this.selectBoolean = false;
        //...
    }
    handleKey(event){
        if (event.type === 'keydown') {
            if (event.code === 'Enter') {
                if (this.selectBoolean) {
                    this.selectBoolean = false
                } else {this.selectBoolean = true}
                }
            }
        }
    get select(){
        return this.selectBoolean
    }
}

```

KeyInput.js

In this code snippet I have added a `selectBoolean` attribute to the `keyInput` class, which is initially set as false. This flag will ensure and provide information determining whether the player has pressed the enter key or not. If the player presses the 'enter' key and `selectBoolean` is not already true, then the `selectBoolean` will be set from false to true.

If the player presses the 'enter' key while `selectBoolean` is true, then the `selectBoolean` will be set from true to false. To code this logic, I will need to make some additional modifications to the `handleKey()` method of the `keyInput` class. Firstly, the code checks if the user is holding down the enter key, and if the `selectBoolean` is already true, is that is the case, it will assume that the player wants to stop interacting with the NPC and it will set the `this.selectBoolean` to false. Otherwise if `selectBoolean` was not true, the method will proceed to set it to true, meaning that the player is willing to interact with another NPC. Furthermore, I also added a getter function that will return the `selectBoolean` attribute easily.

I could have used a different alternative approach common in other existing solutions where the user is required to keep holding the enter key or interaction key in order to interact with the NPC, meaning that they will have to constantly keep holding the enter key, although this approach may be easier to code and more efficient as it won't require additional variables to keep track if the player is in a interaction state or not, this approach may not always be liked by users and stakeholders because it may not be appealing to different users, and it is better to use a toggling system because it allows for more flexibility and requires less input from the user, making the game mechanics much easier.

```

class Game{
  Loop() {
    //...
    const gameLoop = () => {
      //...
      const entities = Object.values(this.map.entities);
      entities.push(player);

      // Iterate through each entity in the list
      entities.forEach(entity => {
        // Initial state
        const state = { map: this.map };

        // add the player's direction and speedBoost to the state
        if (entity.isPlaying) {
          state.direction = this.keyInput.direction;
          state.speedBoost = this.keyInput.speedBoost;
          state.enterBool = this.keyInput.select;
        }
        // passing in the state for every other entity
        entity.update(state);
      });
      //...
    }
  }
}

```

Game.js

When iterating through the map entities array, the game checks to see if any of the entities that is being iterated is the player instance. If it is the player, the state object's properties are set to add more information which are required in the player class.

The value of state.enterBool is set to this.keyInput.select. This.keyInput refers to an input manager in charge of handling user inputs such as key presses. This.keyInput.select returns a boolean value indicating whether or not the user is currently pressing the "enter" key. So, by setting the value of state.enterBool to this.

With keyInput.select, the state object is updated with the current state of the "enter" key, which the player can then use. The map object's **playerInteraction()** method determines whether the player is currently attempting to interact with an NPC or not.

Test

The following test plan will be mainly testing the **playerInteraction()** method and test only 2 different scenarios, the first test to check if this function works by checking if the NPC is interactive when is not moving and the second test to check the functionality of this method when the NPC is moving, meaning it is currently executing a behaviour loop, and the player should be still able to interact with it.

Test No.	What I am testing	Test data	Result Required
1	Player interacting with the standing NPC	playerInteraction()	The NPC should face the player when the player interacts with the NPC as soon as the player presses the enter key. And the player must be frozen and unable to perform other actions
2	Player interacting with a moving NPC	playerInteraction()	The player should be able to interact with an NPC which is part of a behaviour loop.

The `playerInteraction()` function will also be tested to ensure that it performs exactly as expected, this test plan will allow me to ensure that the game will be capable of handling dialogue elements if all of these tests were successful

The first test is a **normal** test which determines whether the game is capable of allowing the player to interact with the NPC. NPC must face towards the player during the interaction stage, meaning as soon as the player presses the enter key. This test is important because I could not test for this aspect of the `playerInteraction()` method, however, if this function does not meet the expected results, then it could restrict and limit the development of the main purpose of the NPC interaction feature which is the dialogue feature, a main component of the game and it must be present as shown in the success criteria.

The second test is a **normal** test which determines whether the game is capable of handling and executing the **playerInteraction()** method effectively when the player tries to interact with an NPC which is moving.

1st Test: First Iteration

```

class keyInput {
    //...
    handleKey(event) {
        //Gets the direction value from input key
        const direction = this.keyDirectionMap[event.code];
        //Gets the index of the direction in the keysHeld array
        const index = this.keysHeld.indexOf(direction);
        //Checks if the user presses a key
        if (event.type === 'keydown') {
            if (event.code === "Enter"){
                console.log("%cThe player is interacting!", 'color: blue')
            }
            else {console.log(event.code)}
            //...
        }
        //...
    }
    //...
}

```

KeyInput.js

This new temporary addition of a conditional statement printing two different console log methods will allow me to debug and test the functionality of this feature, when the enter key is pressed, assuming that I am only press it when I interact with the NPC, the console should print out that I am interacting with the NPC in a different colour. Whereas, if the player presses any other keys, they will be just displayed as the name of those keys, allowing me to check if any other keys which allows the player to perform actions disrupts the interaction stage

1st Test: First Iteration Test Results

The player is interacting!	KeyInput.js:38
20 ShiftLeft	KeyInput.js:40
KeyD	KeyInput.js:40
KeyA	KeyInput.js:40
2 KeyW	KeyInput.js:40
KeyA	KeyInput.js:40
KeyS	KeyInput.js:40
KeyD	KeyInput.js:40
The player is interacting!	KeyInput.js:38
KeyS	KeyInput.js:40

[1st Test: First Iteration Test Video](#)

The test was successful.

The player is unable to move when it is in a state of interaction with the NPC as seen in the video and indicated by the console log.

When the interaction feature is activated, meaning that the console log displays “The player is interacting!”, the movement and transformation keys (WASD keys or ShiftLeft key) do not execute any actions, and only works if the player breaks off from the interactive state.

2nd Test: First Iteration

```
//Creates global Maps object
windowMaps = {
StartingTown: {
    name: "StartingTown",
    entities: { //Collection of entities of StartingHouse map
        npc1: new Obj({
            //sets NPC1 properties
            name: "Boy",
            x: 25, y: 10,
            speed: 1,
            behaviourLoop: [
                { behaviour: "walking", direction: "down", tiles: 1 },
                { behaviour: "standing", direction: "down", time: 1000 },
                { behaviour: "walking", direction: "down", tiles: 1 },
                { behaviour: "standing", direction: "down", time: 1000 },
                { behaviour: "walking", direction: "down", tiles: 1 },
                { behaviour: "standing", direction: "down", time: 1000 },
                { behaviour: "walking", direction: "down", tiles: 1 },
                { behaviour: "standing", direction: "down", time: 5000 },
                { behaviour: "walking", direction: "up", tiles: 4 },
            ],
        }),
    },
    //...
}
```

2nd Test: First Iteration Test Results

2nd Test: First Iteration Test Video

The test was successful.

The interaction feature also works for when the NPC is executing a behaviour loop, where when the interaction state is activated, the behaviour loop is temporarily interrupted until the NPC is not interacting with the player anymore.

Dialogues

Dialogue box

```

class Sprite {
    constructor(config) {
        //...
        // Check if the object type is not monster
        if(this.Obj.type !== "monster"){
            // Creates a image instance
            this.faceset = new Image();
            // Sets the image source
            this.faceset.src = "Characters/" + this.Obj.name + "/Faceset.png"
            // When the image is Loaded
            this.faceset.onload = () => {
                //Then set the flag as true
                this.facesetIsLoaded = true;
            }
        }

        this.dialogue = new Image();
        this.dialogue.src = "HUD/Dialog/DialogBoxFaceset.png"
        this.dialogue.onload = () => {
            this.dialogueIsLoaded = true;
        }
        //...
    }
    //...
}
//...

```

Entity.js

In order to add dialogues and text features which are going to be displayed when the player interacts with an NPC, I will require a dialogueBox where the text and faceset of the NPC will be displayed. There are two image objects that I have created in this class, the first one being the faceset and it is only created if the object's type is not “monster”, as the user won't be able to interact with a monster entity. Then the class loads an image file from the “Characters” folder with the name of the object and “Faceset.png” appended to it. Once the image is loaded, it sets a boolean flag called facesetIsLoaded to true.

The second image object is called dialogue. It loads an image file from the “HUD/Dialog” folder with the name “DialogBoxFaceset.png”. Once the image is loaded, it sets a boolean flag called dialogueIsLoaded to true.

I could have created a new class for the dialogue feature to increase the modularity of my game and simplify the project, however, this would mean that I will need to create a similar class to the sprite class, as it will cause code duplication and increase inefficiencies of the code.

Drawing the dialogue

```

class Obj { //A blueprint for an object in the game
    constructor(config){
        this.text = config.text
        this.greetings = null
    }
    //...
// The drawDialogue method displays text on the canvas
drawDialogue(context, element) {
    // Check if the entity is not a monster
    if (this.type !== "monster") {
        // Draw the dialogue box and face graphic
        context.drawImage(this.sprite.dialogueBox, 0, 90);
        context.drawImage(this.sprite.faceset, 6, 103);
        // Create a container for the text
        const dialogueContainer = document.createElement('div');
        dialogueContainer.className = 'dialogue-container';
        // Create a text element for the dialogue
        const textElement = document.createElement('p');
        // Check if there is already text to display
        if (this.text) {
            textElement.innerText = this.text;
        } else {
            // If there is no text, display a greeting
            if (this.greeting) {
                // If a greeting has already been chosen, use it again
                textElement.innerText = this.greeting;
            } else {
                // Otherwise, choose a new greeting at random and store
                // it in the entity object
                const randomIndex = Math.floor(Math.random() * greetings.length);
                this.greeting = greetings[randomIndex];
                textElement.innerText = this.greeting;
            }
        }
        textElement.className = 'dialogue-text';
        // Add the text element to the dialogue container
        dialogueContainer.appendChild(textElement);
        // Use the parent element of the game canvas as the container
        element.appendChild(dialogueContainer);
    }
}
//...
}

```

Entity.js

I have added two attributes which are set inside: text and greetings.

The text attribute specifies what sentence and information the NPC instance should print, whereas the greetings attribute chooses a random greeting from the following array:

```
window.greetings=
["Hello there! Welcome to our town!",
 "Greetings, traveller! How may I assist you today?",
 "Hi! I hope you're having a great day so far.",
 "Welcome to our humble abode! How can we make your visit more enjoyable?",
 "Hey there! Is there anything I can help you with?",
 "Good day, adventurer! Welcome to our lovely town",
 "Greetings! It's always nice to meet new people. What brings you to our town?",
 "Salutations! I hope you're finding everything you need here.",
 "Hello, hello! I hope you're enjoying your stay in our lovely town.",
 "Hey, stranger! Anything I can do to make your visit more pleasant?"
];
```

Globals.js

Now that I have distinguished these key attributes, I need to code to meet their criteria.

I have implemented a new method, **drawDialogue()** that accepts a context object and an element object as parameters so it can proceed to draw later on. There is a conditional statement within the method to check if the entity is not a monster, as I have explained previously that the player won't be able to interact with a monster type entity.

This method allows for the drawing of the dialogue box as well as a face graphic. It then generates a text element for the dialogue and a container for the text. If the object contains `this.text`, it will be displayed in the text element. If not, it selects a greeting at random from an array called `greetings`, saves it as `this.greeting` for future use, and displays it in the text element. Finally, it appends the text element to the dialogue container and the container to the game canvas's parent element.

```

Loop() {
  const gameLoop = () => {
    // Check if the dialogue container exists
    const dialogueContainer = document.querySelector('.dialogue-container');
    if (dialogueContainer) {
      // If it exists, remove it from the DOM
      dialogueContainer.remove();
    }
    Object.values(entities).forEach(entity => {
      if(entity.interacting){
        entity.drawDialogue(this.context);
      }
    })
  }
}

```

Game.js

The **gameLoop()** function in this code snippet uses the **document.querySelector()** method to determine whether a specific HTML element with the class dialogue-container exists in the DOM. If the element exists, it is assigned to the variable dialogueContainer.

The if statement determines whether the variable dialogueContainer has a value. If it does, it means the dialogue container exists, and the **remove()** method on it is used to remove it from the DOM. This ensures to clear up any existing dialogueBox when the player is not interacting with the NPC anymore.

The second change is that the code iterates through all of the entities in the entities object to the current map and checks if any of these entities are currently being interacted with. If this conditional statement is true, the **drawDialogue()** method for that entity which is interacting with the player.

NPC Dialogue Name

```

class Obj { //A blueprint for an object in the game
constructor(config) {
    this.displayName = config.displayName || "Villager"
}

// The drawDialogue method displays text on the canvas
drawDialogue(context, element) {
    // Check if the entity is not a monster
    if (this.type !== "monster") {
        // Create a container for the text
        const nameContainer = document.createElement('div');
        nameContainer.className = 'name-container';

        // Create a text element for the dialogue
        const nameElement = document.createElement('p');
        nameElement.innerText = this.displayName;
        nameElement.className = 'name-text';
        //...
        // Add the text element to the dialogue container
        dialogueContainer.appendChild(textElement);
        nameContainer.appendChild(nameElement);

        // Use the parent element of the game canvas as the container
        element.appendChild(dialogueContainer);
        element.appendChild(nameContainer);
    }
}

```

Entity.js

I have added a new attribute to the Obj class, **this.displayName** attribute, which stores the value of config.displayName or "Villager" if config.displayName is not provided.

I have also added some new logic to the **drawDialogue() method**, where the updated version generates a container element for the NPC's name text, it creates a nameElement paragraph element with the innerText set to the object's displayName property and the class set to "name-text". The **nameElement** is appended to the **nameContainer**.

I have used the same structure and approach for my previous dialogue box feature.

Finally, both the dialogueContainer and the nameContainer are appended to the element, which is the game canvas's parent element.

```

Loop() {
  const gameLoop = () => {
    // Check if the dialogue container exists
    const dialogueContainer = document.querySelector('.dialogue-container');
    const nameContainer = document.querySelector('.name-container');
    if (dialogueContainer && nameContainer) {
      // If it exists, remove it from the DOM
      dialogueContainer.remove();
      nameContainer.remove();
    }
}

```

Game.js

I also need to remove the name container once the player is done interacting with the NPC. I have already implemented this previously to remove the dialogue text and dialogue box.

CSS Style Dialogue Box

```

/* Dialogue container for displaying dialogue text */
.dialogue-container {
  position: absolute; /* Positioning type */
  top: 70%; /* Distance from top of the screen */
  left: 18%; /* Distance from left of the screen */
  margin-right: 50px; /* Margin on the right side */
}
/* Styling for the dialogue text */
.dialogue-text {
  margin: 0; /* Remove default margin */
  font-size: 40px; /* Set font size */
  word-spacing: 10px; /* Spacing between words */
  font-family: 'NormalFont'; /* Font family */
}
/* Container for displaying character name */
.name-container{
  color: "green"; /* Text color */
  position: absolute; /* Positioning type */
  top: 60%; /* Distance from top of the screen */
  left: 3%; /* Distance from left of the screen */
}
/* Styling for the character name text */
.name-text {
  color: "green"; /* Text color */
  margin: 0; /* Remove default margin */
  font-size: 30px; /* Set font size */
  word-spacing: 10px; /* Spacing between words */
  font-family: 'NormalFont'; /* Font family */
}

```

style.css

I will be required to style and position the dialogues, facesets and the NPC name appropriately, and the only way to achieve this is to use CSS that defines the styles for these elements. The first CSS class, the **.dialogue-container**, which is the element that will store the dialogue box, is positioned using the '**absolute**' property which will position this element relative to the closest positioned ancestor and has a fixed location on the game canvas. The specific fixed position of this CSS class element is 70% from the top of the page and 18% from the left, meaning that this element is positioned at the bottom of the screen, sticking to my approach from the design section after consulting with my stakeholders. Secondly, the **.dialogue-text** class, which is the element for displaying and storing the text that will be displayed when the player interacts with the NPC. I have positioned this element appropriately to fit in the dialogue box with no margins between these two elements, as I set the margin attribute to 0. I have also added custom fonts so that the text dialogues could match the theme of the game, this font embodies the pixelated theme of the game as I have mentioned in the design section.

Test

Test No.	What I am testing	Test Data	Result Required
1	Player interacting with the standing NPC	playerInteraction() drawDialogue()	When the player interacts with any NPC, their respective text or greeting must be displayed onto a dialogue box with the correct CSS styles and positioning.

This test is a **normal** test which determines whether the game is now capable of handling and displaying dialogues with the correct positioning, images, texts and greetings when the player interacts with the NPC. This test is the final test of the project which is also important because I could not test for this feature, however, testing the dialogues will allow me make sure if I have met the requirements and success criteria that I have stated in the design section, as dialogues will allow the user to obtain information from the NPCs, and will also provide for maintenance and possible future development of an implementation of a story mode system to this game, a really common approach in most RPG games.

1st Test: First Iteration

1st Test: First Iteration Test Results



[NPC Interactions & Dialogues video](#)
The test was successful as shown in the video. There's no need to review the development of this feature due to the lack of errors for the player interaction system

Evaluation

Success Criteria

The purpose of assessing that I met the criterias that I set for myself in the analysis section is to assess the effectiveness and determine whether I have met my desired goals for this project. The success criteria provides me with a clear framework for evaluating and identifying the areas of improvements that can be made for future development or maintenance, ensuring that it is effective in helping other developers to understand my game.

Success Criteria	Evidence Requirement	Evidence and Justification
Multiple different maps	Screenshots of different maps	<p>Fully Completed</p> <p><i>Screenshots:</i></p> <p>StartingHouse Map</p> <p>StartingTown Map</p> <p>FishingHut Map</p> <p>Dojo Map</p> <p>The following shows the screenshots of the different maps that I have included in my final game.</p>
Character Movement	Video of the character moving with the right direction input from the user.	<p>Fully Completed</p> <p><i>Video Evidence:</i></p> <p>Player moving video</p> <p>I have shown the player moving around the game with the correct direction and input keys from the user as seen in the console shown in the video.</p>
Speed Boost	Video of the user toggling on speed boost allowing the character to travel at a faster speed	<p>Fully Completed</p> <p><i>Video Evidence:</i></p> <p>Speed Boost Video</p> <p>I have shown a video of the user toggling the speed boost on and off, where the game then responds by changing the speed of the character movement.</p>

Character Walking Animation	Video of the character running through a walking animation with the correct frames.	<p>Fully Completed</p> <p>Video Evidence: Character Walking Animation Video</p> <p>I have shown a video of the character iterating through a correct walking animation. I have developed this function with the aid of the game loop, where it repaints the game canvas continuously at a constant rate, drawing different frames which creates an illusion of motion.</p>
Character Camera	Video of the game where the map moves following the character's movement	<p>Fully Completed</p> <p>Video Evidence: Character camera video</p> <p>I have shown the map moving according to the player's position, providing an illusion that the character is always in the centre of the canvas. Moreover, it shows the correct walking animation depending on the character's direction and behaviour.</p>
Special Ability	A video of the character using their special ability.	<p>Fully Completed</p> <p>Video Evidence: Special Ability Video</p> <p>I have shown the character switch to monster which allows the player to control a different entity and enjoy a faster speed, the user is able to toggle on and off the special ability, which is the transformation ability.</p>
Effects & HUD	A video of the character using their special ability which will show an effect animation and display a HUD.	<p>Fully Completed</p> <p>Video Evidence: Effects & HUD video</p> <p>I have successfully added HUD and effects complementing the character's special ability of transformation as shown in the video. The HUD lights up and off, and the effect animation is displayed when the player toggles between two entities using the transformation ability.</p>

Character Collisions	Video of the character not being able to barge through walls, trees, entities and etc,	<p>Fully Completed</p> <p>Video Evidence: Character Collisions video</p> <p>I have successfully met the criteria requirements for the character collisions. As seen in the video, the character is not able to walk through map objects, walls and entities such as other NPCs.</p>
Customisable player skin	A video of the user changing the character skin.	<p>Partially Completed</p>  <p>I was not able to let the user choose the sprite they are willing to use to play with because I was not successful in adding a menu to the game. However, the player entity can support different sprite skins by manually changing the skin name in the global player object.</p>
Map Switching	Video of the player exploring and switching between different maps while displaying a transition.	<p>Fully Completed</p> <p>Video Evidence: Map Switching video</p> <p>I have successfully met the criteria of exploring maps by allowing the user to travel between maps with an appropriate transition displaying during the process as seen in the video.</p>
Entities in each map	Screenshots of entities for every map	<p>Fully Completed</p> <p>Screenshots: StartingHouse Map StartingTown Map FishingHut Map Dojo Map</p> <p>As seen in the following screenshots, I have added different entities in each map, most NPCs have different sprites and positions in the map. This is important because it allows each map to differentiate from one another and gives a sense to the player that they are exploring different locations, supporting more variety to the game.</p>

NPC behaviours and collisions	A video of the NPCs moving in specific patterns and colliding with the player and other entities.	<p>Fully Completed</p> <p>Video Evidence: NPC behaviours video & NPC collision Video</p> <p>I have successfully met the criteria where the NPCs are able to have their own individual behaviours which is easily configurable and I have added collisions between entities and the player as shown in the two videos I have added as evidence.</p>
NPC Interactions & Dialogues	Video of an entity responding to a character interaction with a dialogue	<p>Fully Completed</p> <p>Video Evidence: NPC Interactions & Dialogues video</p> <p>I have successfully met the criteria where the player is able to communicate with different NPCs by pressing the 'Enter' key while facing and colliding with the NPC that the player is aiming to interact with. Once the player interacts with the desired NPC, that entity freezes their behaviour loop and displays their dialogue, faceset and their name. This allows the player to obtain information, especially useful for adding a story mode to the game for further maintenance.</p>
A battle system	A video of a turn-based battle system where the player is able to attack the opponent	<p>Not completed</p> <p>I have failed to meet the criteria to add a battle system in any form or shape due to the lack of time and I have underestimated the complexity of this feature, my new estimation suggests that the game battle system will take twice as long as the features I have added all combined. Adding a battle system is more like creating a brand new game with different logics, classes and functions.</p>
There should be a menu	A screenshot of a menu window with different options and settings for the user to configure with	<p>Not completed</p> <p>I was unable to meet this criteria also due to the lack of time and the different logic required to build this feature due to my misunderstanding. However, to meet this criteria, I would have needed to write new code without reusing my existing solutions.</p>

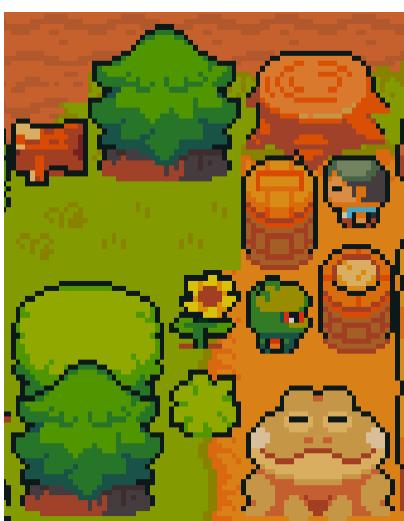
Usability Features

Intuitive Controls: One usability feature that I have added is the easy controls and keys configured to execute certain functions which are easy to understand. This allows the user to focus on the gameplay rather than having trouble figuring out how to perform some actions, which could affect their gameplay experience negatively.

I could add complicated controls, by adding a larger range of controls and the player to perform which could add an additional layer of depth to the gameplay, however, this approach may not always be feasible due to the high steep learning curves that it would cause learning the new game mechanics. This could be an issue especially for new players, casual players and even non-RPG players, causing an uprising in negative user experience and further confusion.

The simple controls that I have implemented in this game are W,A,S,D and the arrow keys for the character's movement, the 'LeftShift' key for transformation ability and finally the 'Enter' key to allow the user to interact with NPCs etc.

In summary there are about 3 main controls for 3 different features, allowing the player to pick up and learn the controls more easily so they can experience the game instead of having trouble with the controls. Furthermore, relating to input keys, I have also added error prevention as a usability feature, where the game ensures other keys except for the one specified is not taken into account and performs unspecified actions. This leads me to believe that this usability feature has been **successfully implemented.**



Consistent design and UI: This usability feature will allow players to recognise and understand the different elements of the game much more easily. I have completely met this criteria by using a one game asset which contains a set of spritesheets, images, and more elements which complement each other to contribute to an overall aesthetic appeal of the game. This also allows the game to be visually coherent and consistent. This could be more immersive and engaging for the user, leading to the creation of a more polished and professional-looking game, which suggests that this usability **feature is successful.**

Dialogues: Another great usability feature that I have implemented in this game is the use of dialogues by interacting with different NPCs. This feature could be really useful for future development, as it will allow and facilitate new features to the game such as adding a game story using dialogues, which is already a common feature in many other existing RPG games such as Pokemon Firered, Legend of zelda etc.



Storytelling can be used to provide important information about the game's story, setting and characters by using NPC dialogues to convey it. The player will experience a more compelling tale overall thanks to this method, which also increases immersion. In addition, this strategy will enable the future development of new features that will deepen the game even further, such as missions, fights, and simply more worldbuilding background. However, I have not added these features because of the mishandling and underestimating the length of this project. The only way to implement these features that uses the dialogue system is in the further development of this game, leading me to believe that this usability feature is **partially successful**.



HUDs (Heads-up display): one of the HUDs that I have used , the transformation HUD, is clearly visible and the design matches the rest of the UI. When the transform ability is toggled on, the HUD obviously 'lights up' by increasing the opacity of it instantly.

Evidence of HUD usability - As you can see the HUD is clear, large and provides a clear visual cue to the player that tells them if they have toggled transformation ability on and off. This is an example of contextual overlay, where the HUD only appears when it is needed. I could use audio cues instead to alert the user that they have toggled the transformation ability, as it helps to create a minimalist game design and reducing visual complexity on the canvas, however, audio cues are not clearer than visual HUDs because it does not provide clear and precise information which may lead to some misinterpretation. Therefore, I believe that HUDs have been **completely successful**.

Performance optimization ensures that the game has been optimised for performance, which means it will enable faster loading times and smoother gameplay, which is especially crucial for low-end devices, which may not necessarily have the same processing capacity as high-end gaming devices. This usability feature addresses one of the project's most fundamental goals, which is to allow all players, particularly those with low-performance devices, to experience smooth gameplay and high performance. I achieved this usability feature by using asynchronous programming, which allows the game to do numerous operations concurrently without stopping the main thread.

```
async fetchCoordinates() {
    // Make a fetch request to retrieve collision data for the map
    const response = await fetch(`/Maps/${this.name}/collision.json`);
    // Parse the response as a JSON object
    const json = await response.json();
    // Assign the first layer of the JSON object to the "walls" property
    this.walls = json.layers[0];
}
```

Map.js - where I have used asynchronous programming

Javascript facilitates the use of asynchronous programming. I have used it for drawing map layers, fetching collisions and other functions which require a lot of executions and traversing data. As seen in the code snippets, I have used '**async**' and '**await**' to let the functions execute at the same time. This is one example of many.

Furthermore, the use of OOP also optimises performance to a certain extent, as this paradigm focuses on reusability, improving code organisation, reducing code duplication etc. As a result, OOP ensures that the game reduces unnecessary code execution, such as minimising the number of loops and executions which allows the game to run faster and allows better performance. However, there could be some constraints to performance as I have used global variables excessively which reduces performance due to the increased memory usage and slower access time. Therefore, I believe this usability feature is partially successful.

Stakeholder Testing

I have allowed most of my stakeholders to test out my game and shown them the requirements and the success criteria that I have met to satisfy them, which would potentially satisfy the larger audience of players that will be able to play my game. I have received a variety of feedback from all the different stakeholders. Here's the following:

Feedback

- 1. Prince Camayah:** "I really enjoyed testing out your game, the graphics were great and the gameplay was engaging. I did notice some bugs with the NPCs while playing, however, so you may want to consider fixing those in future updates or development."
- 2. Muhammin Ahmed:** "As a pixel artist, the way you integrated the graphics with each other in your game was top-notch! The fonts were a bit difficult to read though, so maybe consider making them more clear in the future by maybe changing the font to something clearer."
- 3. Wessel Broek:** "I am a casual gamer, and I discovered yours to be simple to use and enjoy the controls and gameplay. There was one issue, however, causing the game to crash and output an error in the console especially when I run the game for a long time. Other than that, I believe your game is amazing and I will definitely play it occasionally. You might also add sound effects and audio in further development."
- 4. Ivo Igor Kucher:** "I really enjoyed the character movement and animation in your game. The dialogue font was appealing and engaging, I can definitely see how you may be able to turn this into a long story-based game. However, some of the controls were a bit confusing and could use some improvement."
- 5. Andre Billey:** "I thought your game was really fun to play and had a great sense of adventure. I did notice a few spelling errors in the dialogue though, also you may as well provide information about the game mechanics so the new users can perform those actions."
- 6. Bartosz Bester:** "I must say, I was quite impressed with the results and how you met the requirements for the features and other small details. I really enjoyed how the NPCs will have their own behaviour patterns, making them feel more distinguished and special compared to one another, instead of keeping them as static entities on a map. However, I did notice a few bugs with the NPC, as most time when I initiate the game the entities seems to travel within the walls and avoids collision, should definitely fix that but overall I am really impressed with the game and would definitely recommend to RPG enthusiasts"

Overall, the final results seem to be appealing to the stakeholders judging from the feedback I've received from contacting them, the main issue that they had were the minor bugs that I have already talked about in the limitations section, such as minor glitches with NPCs etc.

Functional Testing

[Post-development Test Video](#) - I have completed the **functionality testing and usability testing** in this video all at once allowing me to confirm that the whole game works perfectly and explaining each functionality thanks to the voiceover of the video.

Test No.	Feature to Test	Input	Expected Result	Actual Result
1	Player movement	WASD/arrow keys	Player moves in the correct direction, animation and correct camera.	Successful
2	Collisions	WASD/arrow keys	Player cannot move through walls or entities	Successful
3	Dialogue system	Enter key	Player can initiate dialogue with NPC	Successful
4	Map navigation	WASD/arrow keys	Player can travel to several different maps	Successful
5	Transformation & Speed boost ability	ShiftLeft Key	Player switches to a monster entity and moves at faster speed.	Successful

I have finally tested the functionalities of the main features and a number of important game elements that the user can use; including player movement, collision detection, dialogue system, map navigation, and transformation/speed boost ability, I have provided evidence of all these combined in the post-development test video above, you will also see that all of these test were successful and matched the expected results and have tested these functionalities in various circumstances. These results have aided me in confirming that the game works properly and satisfies the requirements and the stakeholders.

Robustness Testing

[Robustness Testing Video](#) - I have noted down the actual results in the robustness table:

Test No.	Test Description	Test Procedures	Expected Result	Actual Result
1	Test if the game is able to handle interruptions during when the user holds a key and executes a task from outside the game simultaneously	Attempt to hold shift key and then right click to toggle on the usual context menu	The game should be able to detect the interrupt and avoid the “stuck key” issue	Failed When the test procedures are executed appropriately, there seems to be a stuck key error, where the game retains the last input as held down even though it has been released already
2	Test the robustness of the game when running for long periods of time	Leave the game running for at least 10 minutes	The game should work as normal even after running for an extended amount of time	Successful The results are as expected
3	Test the robustness of the game when tested under a unstable internet connection	Simulate network issues using a network limiter at varying different speeds, then test out the game and its functionalities.	The game should be able to handle really slow internet connections due to its small scale	Partially successful Most times the game is able to handle slow internet speed but on rare occasions it may lead to errors with NPC collisions
4	Test the robustness of the game when the website is refreshed at really fast rates and repeatedly	Attempt to spam click the refresh button	The game should load, not crash at all times and initialise itself for every refresh	Partially successful The game does not crash but there are some issues with collisions where it does not load properly only on extremely slow internet speed.
5	Test the robustness when the user spams a key	Spam all the valid keys on the keyboard during the gameplay	The game should work as normal without crashing	Successful The results are as expected

I have collected all the errors and limitations from all the testing methods that I have used and came up with proposed solutions for further development and analysed the causes of this issue in the following section:

Limitations & Issues

Lack of clear instructions: One of my project's primary disadvantages is the absence of clear instructions. Most games provide concise and straightforward instructions to the user at the outset of gameplay on controls, tooltips, and other game mechanics guides. This issue was brought up from the stakeholder testing feedback. Players may find it difficult to use the game mechanics, learn and figure out how to perform different actions that are available due to the lack of instructions in the game.

I could use NPC dialogues, however, to display instructions and important information that the player may require. This only involves changing the dialogue of any NPC in the Global.js file. Here's the solution for the previous example using this approach:



This approach is a common use in other similar games and gives a clear purpose to NPCs, as now the player can communicate and interact with these NPCs to gather more information about the instructions, this also encourages the user to interact with the game more to fulfil their curiosity of finding out new information and features. This approach was suggested by one of my stakeholders during the stakeholder testing feed (5th stakeholder). This solution works because, for example, if a new user is trying out this game for the first time, they wouldn't be aware of the transform ability feature, which requires them to press shift. However, using dialogues will solve this problem, as shown in the screenshot above, one of the NPC can let the user know they can press the LeftShift key to activate the transform ability.

I could alternatively use a different approach such as adding instructions to the start screen when the user initiates the game which is also a common approach to this limitation, however, that would break immersion, unnecessary clutter of information and require additional code because it is not using the existing solutions. Using cues instead is far more effective than adding instructions at start because it encourages exploration and discovery, especially for an exploration-based game like mine.

Lack of customisable options: As mentioned previously, due to the lack of a menu feature, where the user is allowed to configure the settings of the game, I am unable to allow the user to customise their character sprite. The only way to solve this limitation is to code a menu class in future development.

Excessive use of Global variables: This may lead to a decrease in the performance of the game as the excessive use of global variables in the game will force the game to increase its memory usage, leading to slower loading times, especially for global objects as most of the main components and objects such as the map classes, NPCs and player are all stored a global variables. This may also lead to some constraints in future development because it reduces flexibility and may make the game less scalable. However, I could argue that the global variables that I am using are instances, therefore I could still update and change their attributes. There are some solutions for this limitation indeed, the common approach would be to use function closures, which create private variables that are not accessible outside the function scope, this is a form of encapsulation. For example, In future development I could create a factory of function that returns an object with a public interface and private state.

Held Enter Key limitation: If the enter button is held repeatedly when the player interacts with another NPC, the dialogue will display on and off really fast as seen in the following video: [Evidence of dialogue limitation](#) - This limitation can be distracting to the player if they constantly hold the enter key and they may think that the game is broken, this limitation is a major flaw in this game because it does not handle user key inputs appropriately. However, to fix this issue in future development, I could add a delay function between when the user interacts with an NPC. The best way to implement this solution would be to use a timer to prevent rapid triggering of the function when the Enter key is constantly held. I could use a debounce library, which would mean there could require less code to implement this solution, however, most debounce libraries are not supported on different browsers which could lead to compatibility issues and poor performance, and also restrict some users to play with their desired browsers.

NPC Bugs: I have detected a lot of limitations from the post-development testing especially with robustness testing and stakeholder testing. The main limitation that I have seen from the NPCs that were initialised and heading towards a wall, is that the entity would travel through the walls without any form of collision. I could add some sort of optimisation when it comes to loading collisions and NPC behaviours loops in further development to fix this issue, however, this issue only occurs at extreme conditions such as 512 Kbps which is not even stable for even web browsing, and furthermore this is out of the requirement range that I have imposed in the hardware requirement section, where it states that the minimum requirement for both the download and upload speed.

Too many requests: One rare issue that I have seen and received feedback from some of my stakeholders (3rd Stakeholder feedback in stakeholder testing section) is that the game would freeze on really rare occasions and output the following error message. This issue was brought up when the stakeholder spent way too long on the game without refreshing or any like that. This is a serious issue for maintenance because it could restrict further development of more features that will make the gameplay longer such as story mode, game battle etc.

```

Failed to load resource: the server SpriteSheet.png:1 ↗
responded with a status of 429 ()
Uncaught DOMException: Failed to execute Entity.js:96
'drawImage' on 'CanvasRenderingContext2D': The
HTMLImageElement provided is in the 'broken' state.
    at Sprite.drawObj (https://2d-rpg-adventure-game-code.caviohossain.repl.co/Entity.js:96:15)
    at https://2d-rpg-adventure-game-code.caviohossain.repl.co/Game.js:59:25
    at Array.forEach (<anonymous>)
    at gameLoop (https://2d-rpg-adventure-game-code.caviohossain.repl.co/Game.js:58:33)

```

This issue suggests that the server is requesting too many requests, specifically in drawing all the different entities in one map as deduced from the error line: '**Entity.js:96**' as seen in the code error snippet above, this line of code draws the sprite of the entities. The game freezes when the server is limiting the number of requests within a certain time period, preventing some entities from being loaded and drawn onto the canvas, leading to some gaps in the code, which finally leads to the game breaking and freezing.

There is a common solution for this limitation which can be implemented in future solutions, one approach to prevent this error is to check the source of the sprite image that the game is going to draw onto the canvas, making sure the source is valid. This approach will involve using the onerror event on the HTMLImageElement object to handle this error.

I could just use a try-catch block and handle the error in the catch block, which prevents the code from crashing, however, it does not provide any feedback to the user as to why some entities are not being displayed. Whereas my original approach allows a placeholder entity (which is guaranteed to be loaded and drawn) to be drawn onto the coordinate where there is the error. Therefore, my original approach is far more superior than using a try-catch block algorithm instead.

Maintenance

I have coded the entirety of this project using Object-Oriented Programming construct, which involves organising class hierarchies, ensuring class focus and abstraction principles. Clear inheritance relationships should be maintained while avoiding code bloat, which is a situation where the code becomes excessively complex and difficult to maintain. This happens when developers add too many features, however, OOP would solve this issue as it allows the code to be well-organised and clear thanks to its inheritance principles.

I have used the GitHub version control to maintain the project . At the start of this project, I have created a new repository to store all my code, this allowed me to commit every change to the code which enabled me to track my progress, revert any changes and eased the testing process. Additionally, I have used branches to ensure further maintenance. Branches allows the developers to work on new features without disrupting the main codebase.

This is particularly valuable because it enables the addition of new features without altering the codebase during further development. The owner of the repository must approve the addition of new features using a pull request before determining whether the changes are compatible with the real code. I can proceed and merge the new features to the main branch if the process is successful and the new code is compatible. However, if the new branch is not compatible or not needed, worse case scenario, the new branch feature can be just deleted. Overall, this prevents issues, keeps the stability of the main branch and ensures the code is maintained properly.

Furthermore, I have written this code on the Replit IDE, which offers an all-in one platform that includes many features to ease development such as terminals, code editors and a version control system using the previously mentioned Github, ensuring the effectiveness of maintaining the projects as developers can easily fork, edit, test and deploy code all from one place, facilitating future development. Replit also offers backups of code changes which ensure the project is never accidentally deleted.

The use of global variables, which are easily overwritten and updated in further development, could cause maintenance problems and errors in my game.

Global variables can make it challenging to modularise the codebase, which makes it more challenging to manage changes to the code over time, especially when the codebase expands and gets more complex in future development. Limiting the use of global variables and switching to alternatives such encapsulating the variables within functions when necessary can, however, fix and prevent this maintenance issue. By reducing the possible impact of overwrites, this strategy makes the project more modular and easier to maintain.

Further Development

There were one or two success criterias that I have missed because of my underestimation of the size, complexity and difficulty of this project, especially when I am new to game development and new to using JavaScript, as a result I ended up extending my development process by a lot because of the variety of errors, time spent looking for solutions and even trying to understand how to approach certain problems or the development of some features. However, If the game could have been further developed, I would have added a game battle feature, as I was planning to do so either way to my proposed solution, as it would have added an extra layer of difficulty, added a competitive aspect that can improve improvement and increased the complexity of my game rather than just being a exploration open-world game. I didn't manage to incorporate this crucial component into the game because of a lack of time and an incorrect estimate of the project's length. Due to the amount of sub-features the game battle system will require, if I had been successful in adding a game battle feature, it would have taken much longer and increased the development time significantly.

Moreover, introducing a game battle system would be equivalent to developing an entirely new game within a game. Another feature that I had intended but was unable to implement, code, and test on time due to the length of the development process;

I could have incorporated player sprite customisation, which could have added depth of individuality to the game, allowing the player to feel more engaged with the game. However, although this feature is partially completed, because the player sprite is easily configurable by changing their 'name' attribute, it would have required the development of a menu screen, which I was also planning to develop but did not have enough time, making the development of player sprite customization impossible.

There are many features in my game such as the dialogue system, which could facilitate further development of new cool features such as a story mode system. This feature would provide an exciting element to the game and make it feel more engaging, half of the story mode feature is already developed and tested in the current solution, which is the dialogue system. The other half involves developing cut scenes, which won't be much difficult to implement as I already have logic in my game to freeze entities, specific configurable movement behaviours and so much more. I could have added this to the current solution or even planned it in the design section, however, it would have removed the element of the game being an open-world game and taken away that freedom, moreover, in the stakeholders features survey, most stakeholder voted for an open-world game rather than a progress with story type of game.

Furthermore, I could have improved and expanded on existing features such as adding additional maps, with more entities and objects which is easily manageable thanks to the use of OOP, however, this does not increase complexity, but rather the size and scope of the game which could make the game more interesting and engaging, and encourages exploration which is the main objective of this game, therefore, adding more maps is the easiest way to make my game better in further development.

Final Code

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>2D RPG</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
    <link rel="preload" href="HUD/Font/NormalFont.ttf" as="font" type="font/ttf"
crossorigin>
</head>

<body>
    <!-- The game div is going to contain the display of the game -->
    <div class="game">
        <!-- The game canvas -->
        <canvas class="canvas"></canvas>
    </div>

    <!-- Scripts -->
    <!-- Links all the scripts back to the HTML file-->
    <script src="Game.js"></script>
    <script src="MapClass.js"></script>
    <script src="Entity.js"></script>
    <script src="HUD.js"></script>
    <script src="Player.js"></script>
    <script src="KeyInput.js"></script>
    <script src="Globals.js"></script>
    <script src="init.js"></script>

    </body>
</html>
```

Index.js

```

// A game parent class which include every component of the game
class Game {
  constructor(config) {
    //HTML tag where it will display the game
    this.element = config.element;
    //Reference to the HTML canvas
    this.canvas = this.element.querySelector(".game canvas");
    //Will give us access to different drawing methods
    this.context = this.canvas.getContext("2d");
    //The current map that the user is playing on
    this.map = null;
  }

  Loop() {
    let fadeIn = false; //Declare and set the fadeIn transition to false initially
    let fadeInOpacity = 0; //Declare and set opacity to 0 initially
    //Declares the game Loop
    const gameLoop = () => {
      if (!fadeIn) { //If the game is not in the fadeIn transition process
        // Clears the canvas
        this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);

        // Check if the dialogue container exists
        const dialogueContainer = document.querySelector('.dialogue-container');
        const nameContainer = document.querySelector('.name-container');
        if (dialogueContainer && nameContainer) {
          // If it exists, remove both elements from the DOM
          dialogueContainer.remove();
          nameContainer.remove();
        }
      }

      // Create a list of entities from the values of the entities in the map object
      const entities = Object.values(this.map.entities);
      entities.push(player); //Add the player to the list
      // Iterate through each entity in the list
      entities.forEach(entity => {
        // Initial state variable with the map instance
        const state = { map: this.map };
        // add the player's direction and speedBoost to the state
        if (entity.isPlayer) {
          state.direction = this.keyInput.direction;
          state.speedBoost = this.keyInput.speedBoost;
        }
        // passing in the state for every other entity
        entity.update(state);
      });

      //Draw Lower and collision Layers
      this.map.drawLower(this.context, player);
      this.map.drawCollision(this.context, player);
    }
  }
}

```

```

//For every entity available in the current map...
Object.values(entities).forEach(entity => {
    //Draw each entity by calling its method
    entity.sprite.drawObj(this.context, player);
})

//For every effect available...
Object.values(player.fx).forEach(effect => {
    //Draw the effect by calling its method
    effect.drawFX(this.context, player)
})

//Draw Upper Layer
this.map.drawUpper(this.context, player);

//For every HUD available...
Object.values(player.hud).forEach(hud => {
    //Draw the HUD by calling its method
    hud.drawHUD(this.context)
})

//Check if any entity is interacting with the player
Object.values(entities).forEach(entity => {
    //If an entity is interacting...
    if (entity.interacting) {
        //Display the dialogue
        entity.drawDialogue(this.context, this.element);
    }
})

// Check for player exiting or entering a new map
Object.entries(this.map.exits).forEach(([key, exit]) => {
    if (player.x / 16 === exit.x && player.y / 16 === exit.y) {
        // Start the fade in transition
        fadeIn = true;
        fadeInOpacity = 0;
        //Change to the respective map
        this.map = window.mapDict[this.map.update(key, exit)];
        //Traverse the collision json file for the new map
        this.map.fetchCoordinates();
    }
});
}
}

```

```

// If fade in is true
if (fadeIn) {
    //The fade-in transition colour
    this.context.fillStyle = "black";
    //The level of opacity
    this.context.globalAlpha = fadeInOpacity;
    //Make sure it covers the whole canvas
    this.context.fillRect(0, 0, this.canvas.width, this.canvas.height)
    //Keep increasing the opacity of the transition
    fadeInOpacity += 0.025;
    //When opacity reaches a certain value...
    if (fadeInOpacity >= 0.6) {
        // Then If opacity has reached the opacity limit, stop the fadeIn
        fadeIn = false;
        //Fully change to black screen
        this.context.globalAlpha = 1; //Last step of the transition process
    }
}
//Move on to the next frame
requestAnimationFrame(gameLoop);
};

//Call the game Loop method again
gameLoop();
}

//This method initialises the necessary methods, instances and the game Loop
init() {
    //Select the starting map
    this.map = mapDict.StartingHouse;
    //Initialise the collisions first then...
    this.map.fetchCoordinates().then(() => {
        this.keyInput = new keyInput(); //Create a keyInput instance
        this.keyInput.init(); //Initialize the keyInput instance
        this.Loop(); // Initialise the game Loop
    });
}
}

```

Game.js

```

//declare class Map with constructor and drawLayers method
class Map {
  constructor(config) {
    //Assigns a name for the map
    this.name = config.name;
    this.walls = config.walls || {};
    this.exits = config.exits || {};

    //Creates Layer instances and assigns Layer sources
    this.lowerLayer = new Image(); //Lower Layer
    this.lowerLayer.src = "/Maps/" + this.name + "/lower layer.png";

    this.collisionLayer = new Image(); //Collision Layer
    this.collisionLayer.src = "/Maps/" + this.name + "/collision layer.png";

    this.upperLayer = new Image(); //Upper Layer
    this.upperLayer.src = "/Maps/" + this.name + "/upper layer.png";

    //Stores all the entity instances of the map
    this.entities = config.entities;
  }

  //draw Lower Layer image
  drawLower(context, camera) {
    //Each Layer image instance is drawn and taking into account the camera and player's position
    context.drawImage(this.lowerLayer, 9 * 16 - camera.x, 4 * 16 - camera.y);
  }

  //draw collision Layer image
  drawCollision(context, camera) {
    context.drawImage(this.collisionLayer, 9 * 16 - camera.x, 4 * 16 - camera.y);
  }

  //draw Upper Layer image
  drawUpper(context, camera) {
    context.drawImage(this.upperLayer, 9 * 16 - camera.x, 4 * 16 - camera.y);
  }

  //Method for executing interactions between the player and the NPCs
  playerInteraction(state) {
    // Define a dictionary that maps the direction string to the corresponding x and y offsets
    let directionDict = {
      "up": [0, -1], "down": [0, 1],
      "right": [1, 0], "left": [-1, 0]
    };

    //New x and y coordinates of the next tile in front of the player
    const newX = player.x / 16 + directionDict[player.direction][0];
    const newY = player.y / 16 + directionDict[player.direction][1];
  }
}

```

```

// Initialize a flag to indicate whether the player is facing an entity...
//...that is not a monster
let isFacingPlayer = false;
// Iterate over all the entities
Object.values(this.entities).forEach(entity => {
    // Check if the entity is not a monster
    if (entity.type !== "monster") {
        // Check if the entity is located in front of the player
        if (entity.x / 16 === newPlayerX && entity.y / 16 === newPlayerY) {
            //If the player has inputted enter...
            if (selectBoolean) {
                //... Then set the entity as interacting and freeze the player
                entity.interacting = true;
                player.freeze = true;
            } else {
                //Otherwise set them to false and unfreeze the player
                entity.interacting = false;
                player.freeze = false;
            }
            // Set the flag to indicate that the player is facing an entity
            isFacingPlayer = true;
        }
    }
});
// Return whether the player is facing an entity that is not a monster
return isFacingPlayer;
}

checkCollision(obj) {
    // Set initial value of collision as false
    let collide = false;
    // Define a dictionary of direction with corresponding x and y values
    let directionDict = {
        "up": [0, -1], "down": [0, 1],
        "right": [1, 0], "left": [-1, 0]
    };
    // Destructure x and y values from the dictionary based on the input direction
    const [xValue, yValue] = directionDict[obj.direction];
    // Update x and y values based on direction
    const x = Math.round(obj.x / 16 + xValue);
    const y = Math.round(obj.y / 16 + yValue);
    // Check if new x and y values are within the bounds of the walls
    if (x >= 0 && x < this.walls.width && y >= 0 && y < this.walls.height) {
        // Check if there are any walls defined
        if (Object.keys(this.walls).length !== 0) {
            // Check if the current position contains a wall
            if (this.walls.data[x + (y * this.walls.width)] !== 0) {
                collide = true;
            }
        }
    };
}

```

```

// Check for collision with entities, except the player
Object.values(this.entities).forEach(entity => {
    //Player to entity collision
    if (x == Math.round(entity.x / 16) && y == Math.round(entity.y / 16)) {
        collide = true
    }
    //Entity to player collision
    if (x == Math.round(player.x / 16) && y == Math.round(player.y / 16)) {
        collide = true
    }
    // Check for player-to-entity collision while both are moving
    if (obj.isPlayer && entity.behaviour === "walking") {
        // Calculate the new position of the entity based on its current direction
        const [entityXValue, entityYValue] = directionDict[entity.direction];
        const newEntityX = entity.x / 16 + entityXValue;
        const newEntityY = entity.y / 16 + entityYValue;
        // Check if the player's position falls within the entity's new and current positions
        if ((x >= newEntityX && x <= entity.x / 16) || (x >= entity.x / 16 && x <= newEntityX))
    {
        if ((y >= newEntityY && y <= entity.y / 16) ||
            (y >= entity.y / 16 && y <= newEntityY)) {
            // If so, set the collide flag to true
            collide = true;
        }
    }
}
});

// Check for entity-to-player collision while both are moving
if (!obj.isPlayer && player.behaviour === "walking") {
    // Calculate the new position of the player based on its current direction
    const newPlayerX = Math.round(player.x / 16 + directionDict[player.direction][0])
    const newPlayerY = Math.round(player.y / 16 + directionDict[player.direction][1])
    // Check if the entity's position matches the player's new position
    if (x == newPlayerX && y == newPlayerY) {
        // If so, set the collide flag to true
        collide = true;
    }
}
return collide;
}

async fetchCoordinates() {
    // Make a fetch request to retrieve collision data for the map
    const response = await fetch(`/Maps/${this.name}/collision.json`);
    // Parse the response as a JSON object
    const json = await response.json();
    // Assign the first layer of the JSON object to the "walls" property
    this.walls = json.layers[0];
}

```

```

update(key, exit) {
    player.x = exit.newX * 16;
    player.y = exit.newY * 16;
    return Object.keys(window.mapDict).
        find(k => window.mapDict[k].name === exit.name)
}
}

```

Map.js

```

class HUD {
    constructor(config) {
        this.name = config.name //Sets the name used for filepath
        this.type = config.type //Sets the type of HUD it is

        //Coordinate position of where to draw this HUD
        this.x = config.x;
        this.y = config.y;

        //Sets the height and width of the HUD
        this.height = config.height;
        this.width = config.width;

        this.opacity = config.opacity || 1; // set default opacity to 1 if not provided
    }

    drawHUD(context) {
        if (this.type === "transform") { //If the type of the HUD is a transform HUD
            this.transformHUD = new Image(); //Create a new image instance for transformHUD
            //Set the transform HUD default background
            this.transformHUD.src = "HUD/NinePathRect/DialogueBubble2.png"
            // Create a new image for the monster element
            this.image = new Image();
            //Set the source of the image using the name for filepath
            this.image.src = "Monsters/" + this.name + "/SpriteSheet.png"
            context.globalAlpha = this.opacity; // set the global alpha value
            context.drawImage(this.transformHUD, 8, 8) //Draws the transform HUD background
            //Draws the monster element taken from the spritesheet
            context.drawImage(this.image, 0, 0, 16, 16, 16, 16, 16, 16)
            context.globalAlpha = 1; // reset the global alpha value back to 1
        }
    }
}

```

```

class FX {
  constructor(config) {
    this.skin = new Image(); // Create a new Image object for the FX sprite
    this.name = config.name // Set the name used for the filepath
    this.skin.src = "FX/" + config.name + "/SpriteSheet.png" // Set the file path for the sprite
    this.isFinished = config.isFinished || false; // Sets a flag to control the animation loop
    this.skin.onload = () => { // When the sprite is Loaded, mark as Loaded
      this.isLoaded = true;
    }
    this.updated = false;
    this.animationsMap = config.animationsMap || { // Create an animations map or use a default
      "play": [[1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0]]
    }
    this.animationSet = config.animationSet || "play"; // Set the initial animation set or use a default
    this.currentSpriteFrame = 0; // Set the initial sprite frame
    this.framesLimit = 2; // Set the number of frames to wait between animation frames
    this.framesLeft = this.framesLimit; // Set the frames left to the frames limit
  }
  updateFramesLeft() {
    if (this.framesLeft > 0) { // If there are frames left...
      // decrement frames left and return
      this.framesLeft--;
      return;
    }
    this.framesLeft = this.framesLimit; // Reset frames left
    this.currentSpriteFrame += 1; // Increment the current sprite frame
    // If the current sprite frame doesn't exist in the animations map...
    if (this.animationsMap[this.animationSet][this.currentSpriteFrame] === undefined)
      this.currentSpriteFrame = 0; // reset to 0
      this.isFinished = true; and mark as finished
      return;
    }
  }
  drawFX(context, camera) {
    // Calculate the coordinate position of the effect taking into account the camera
    const x = player.x + 9 * 16 - camera.x
    const y = player.y + 4 * 16 - camera.y
    // Only update the animation if it's not finished and the sprite is loaded
    if (this.isLoaded && !this.isFinished) {
      // Get the coordinate position of the current sprite frame
      const fx = this.animationsMap[this.animationSet][this.currentSpriteFrame][0] * 16
      const fy = this.animationsMap[this.animationSet][this.currentSpriteFrame][1] * 16
      // Draw the sprite
      context.drawImage(this.skin, fx, fy, 16, 16, x, y, 16, 16)
      this.updateFramesLeft() // Update the frames left
    }
  }
}

```

```

class Player extends Obj { //GameObj that can be controlled by the user
constructor(config) {
    super(config); //Inherits methods and attributes from Obj
    //Determines whether this instance is the player or not
    this.isPlaying = true; //Since this is the player class, this attribute will be true
    //Determines how many tiles it has to travel
    this.tilesLeft = 0 //Initially, its set to 0
    //Position coordinates in pixels
    this.x = config.x * 16 || 4 * 16; //If not provided, default to (4,4)
    this.y = config.y * 16 || 4 * 16;
    //Contains the sprite for default character sprite
    this.originalSprite = {
        name: this.name,
        type: this.type
    }
    //Contains the sprite for the transform entity
    this.transform = config.transform

    //Sets the player's HUD that are available to use
    this.hud = config.hud
    //Initialise the transformHUD using the transform sprite name for the file path
    this.hud.transformHUD.name = this.transform.name
    //Sets the player's effects that are available to use
    this.fx = config.fx

    //Assigns the axis and the value for the corresponding direction
    this.directionDict = {
        "up": ["y", -1], "down": ["y", 1],
        "right": ["x", 1], "left": ["x", -1]
    }
    //Determines whether the player is able to move or not
    this.freeze = false; //Initially, don't freeze the player
}

//Updates the character in each loop
update(state) {
    // If the player has pressed 'enter' key
    if (selectBoolean) {
        // ... Then check if the player is facing an entity that is not a monster
        if (!state.map.playerInteraction(state)) {
            // If not, then set selectBoolean to false
            selectBoolean = false;
        }
    }
    // Call the playerInteraction function with the state parameter
    state.map.playerInteraction(state);
    // If the player is frozen, update the sprite and return
    if (this.freeze) {
        this.sprite.updateSpriteSet("idle-" + this.direction);
        return;
    }
}

```

```

//Update the sprite and position of the player by calling the following methods
this.updateSprite(state)
this.updatePos(state);
//If there are no more tiles left to travel...
if (this.tilesLeft === 0) {
    //if speedBoost is true and the player's speed is default
    if (state.speedBoost && this.speed === 1) {
        //...Then give the player a speed boost
        this.speed = 2; //Boosted running speed
        this.sprite.obj = this.transform //Switch to transformed entity
        this.hud.transformHUD.opacity = 0.8 //Increase the opacity of transformHUD
        this.fx.transformFX.isFinished = false //Activate the transform effect
        //Otherwise, if speedBoost is false and the player's speed is boosted
    } else if (state.speedBoost === false && this.speed === 2) {
        //...Then return player speed to default
        this.speed = 1; //Normal walking speed
        this.sprite.obj = this.originalSprite //Return to original entity
        this.hud.transformHUD.opacity = 0.3 //Reduce the opacity of transformHUD
        this.fx.transformFX.isFinished = false //Activate the transform effect
    }
}

//When TilesLeft is 0 and a direction key has been inputted
if (this.tilesLeft === 0 && state.direction) {
    //Update the direction
    this.direction = state.direction;
    //If there is a collision in front of the player...
    if (state.map.checkCollision(this)) {
        //...Then prevent it from moving
        this.tilesLeft = 0; //Set tiles Left to travel to 0
        this.behaviour = "standing" //Set the behaviour to standing
    } else { //Otherwise...
        this.tilesLeft = this.speed * 16; //Let it move with the desired speed
        this.behavior = "walking"; //Set the behaviour to walking
    }
}
}

updatePos(state) {
    if (this.tilesLeft > 0) { //If the player has to move
        //Checks which direction it needs to move to
        const [axis, value] = this.directionDict[this.direction]
        //Changes their position value on the correct axis
        this[axis] += value * this.speed
        // If this method continues to run, the method will stop when the TilesLeft is 0
        this.tilesLeft -= this.speed * this.speed;
    }
}
}

```

```

class keyInput {
  constructor() {

    // Initialize an empty array to track held keys
    this.keysHeld = [];

    this.speedBoolean = false;

    // Define a map that associates key codes with directions
    this.keyDirectionMap = {
      "ArrowUp": "up",
      "ArrowDown": "down",
      "ArrowLeft": "left",
      "ArrowRight": "right",

      "KeyW": "up",
      "KeyS": "down",
      "KeyA": "left",
      "KeyD": "right"
    };
  }

  init() {
    //The function triggers when the user presses and releases a key, respectively
    document.addEventListener('keydown', this.handleKey.bind(this));
    document.addEventListener('keyup', this.handleKey.bind(this));
  }

  handleKey(event) {
    //Gets the direction value from input key
    const direction = this.keyDirectionMap[event.code];
    //Gets the index of the direction in the keysHeld array
    const index = this.keysHeld.indexOf(direction);
    //Checks if the user presses a key
    if (event.type === 'keydown') {
      //If the held key is not in the keysHeld array...
      if (!this.keysHeld.includes(direction) && direction) {
        //...Then add the direction to the front of the keysHeld array
        this.keysHeld.unshift(direction);
      }
      //If the held key is a Shift button and the speedBoolean is false...
      if (event.code === 'ShiftLeft') {
        //... Then set the speedBoolean as true
        this.speedBoolean = true;
      }
      if (event.code === 'Enter') {
        if (selectBoolean) {
          selectBoolean = false
        } else {selectBoolean = true}
      }
    }
  }
}

```

```

//Checks if the user releases a key
else if (event.type === 'keyup') {
    //If the released key is in the keysHeld array...
    if (index > -1 && direction) {
        //...Then remove the direction of the corresponding key
        this.keysHeld.splice(index, 1);
    }
    //If the released key is a Shift button and the speedBoolean is true...
    if (event.code === 'ShiftLeft' && this.speedBoolean === true) {
        //... Then set the speedBoolean as false
        this.speedBoolean = false;
    }
}
}

//This getter fetches the latest pressed key
get direction() {
    return this.keysHeld[0];
}

//This getter fetches whether the player is running or not
get speedBoost() {
    return this.speedBoolean;
}

//This getter fetches whether the player has pressed 'enter' or not
get select() {
    return this.selectBoolean
}
}

```

KeyInput.js

```

//Creates a global object for storing all the HUD
const playerHUD = {
  transformHUD: new HUD({
    type: "transform",
    opacity: 0.5,
  }),
  heartHUD: new HUD({
    src: "HUD/Heart.png",
    opacity: 1
  })
}

//Creates a global object for storing all the effects
const playerEffects = {
  transformFX: new FX({
    name: 'Shield',
    isFinished: true
  })
}

//Creates a global object for initialising the player instance
window.player = new Player({
  x: 25, y: 15, //Initial position coordinates
  //Sets the name for filepath of spritesheets and faceset
  name: "GreenNinja",
  transform: { //Creates the transform entity object
    name: "GoldRacoon", //Sets the name for filepath of spritsheet
    type: "monster" // Sets the type of the entity for initialising the monster
  },
  hud: playerHUD, //Sets the hud attribute as the global playerHUD object
  fx: playerEffects //Sets the effecs attribute as the global playerHUD object
});
//Sets the initial selectBoolean to false
window.selectBoolean = false

//Creates global Maps object
windowMaps = {
  //creates StartingHouse map
  StartingHouse: {
    name: 'StartingHouse',
    entities: { //Collection of entities of StartingHouse map
      //Creates new entity instances for the map
      npc1: new Obj({
        //Sets the name for filepath of spritesheets and faceset
        name: "OldWoman",
        displayName: "Mum", //Sets the name for dialogue system
        x: 9, y: 3, //Sets the coordinates in for this entity in the StartingHouse map
        direction: "left", //Sets the direction it is facing
        //Sets the dialogue text to display
        text:"When are you going to stop playing around and get a real job?"
      })
    }
  }
}

```

```

},
//Stores all the possible ways to exit and enter the map
exits: {
  StartingTown: {
    name: "StartingTown", //Stores name of the exit map
    x: 7, y: 7, //Stores the coordinates from StartingHouse to enter StartingTown
    newX: 28, newY: 9 //Stores the coordinates from StartingTown to enter StartingHouse
  },
},
},
StartingTown: {
  name: "StartingTown",
  entities: { //Collection of entities of StartingHouse map
    npc1: new Obj({
      //sets NPC1 properties
      name: "Boy",
      x: 25, y: 10,
      speed: 1,
      behaviourLoop: [
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 1000 },
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 1000 },
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 1000 },
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 5000 },
        { behaviour: "walking", direction: "up", tiles: 4 },
      ],
      text:"Hey, you look like you could use a good time - I know all the best spots in this town, if you're interested."
    }),
    npc2: new Obj({
      name: "MaskFrog",
      x: 16, y: 11,
      behaviourLoop: [
        { behaviour: "standing", direction: "up", time: 8000 },
        { behaviour: "standing", direction: "right", time: 3000 },
      ],
      text: "Ribbit, ribbit, just taking a break from being a ninja, nothing to see here."
    }),
    npc3: new Obj({
      name: "OldMan",
      x: 34, y: 11.5,
      behaviourLoop: [
        { behaviour: "standing", direction: "down", time: 9000 },
        { behaviour: "standing", direction: "right", time: 9000 },
        { behaviour: "standing", direction: "up", time: 9000 },
        { behaviour: "standing", direction: "left", time: 9000 }
      ]
    })
}

```

```

}),
npc4: new Obj({
  name: "Monk2",
  x: 39, y: 15,
}),
npc5: new Obj({
  name: "EskimoNinja",
  x: 33, y: 7,
  text: "Oi, bruv, you better not be stepping in my turf fam."
}),
npc6: new Obj({
  name: "Princess",
  x: 19, y: 18,
  direction: "right",
  behaviourLoop: [
    { behaviour: "walking", direction: "up", tiles: 1 },
    { behaviour: "standing", direction: "up", time: 9000 },
    { behaviour: "standing", direction: "down", time: 5000 },
    { behaviour: "walking", direction: "down", tiles: 2 },
    { behaviour: "standing", direction: "down", time: 9000 },
    { behaviour: "standing", direction: "up", time: 5000 },
    { behaviour: "walking", direction: "up", tiles: 1 },
  ],
  text: "Good day to you, may your endeavors be blessed with success and prosperity."
}),
npc7: new Obj({
  name: "Villager",
  x: 31, y: 8,
  speed: 2,
  behaviourLoop: [
    { behaviour: "walking", direction: "left", tiles: 1 },
  ]
}),
monster1: new Obj({
  name: "Racoon",
  x: 23, y: 16,
  type: "monster",
  speed: 1,
  behaviourLoop: [
    { behaviour: "walking", direction: "right", tiles: 4 },
    { behaviour: "walking", direction: "down", tiles: 3 },
    { behaviour: "walking", direction: "left", tiles: 4 },
    { behaviour: "walking", direction: "up", tiles: 3 },
  ]
}),
monster2: new Obj({
  name: "Dragon",
  x: 27, y: 21,
  type: "monster",
  direction: "down",
  speed: 2
}
)

```

```

        }),
    },
    exits: {
      Dojo: {
        name: "Dojo",
        x: 36, y: 15,
        newX: 8, newY: 16
      },
      StartingHouse: {
        name: 'StartingHouse',
        x: 28, y: 8,
        newX: 7, newY: 6
      },
      FishingHut: {
        name: 'FishingHut',
        x: 35, y: 6,
        newX: 3, newY: 32
      },
    },
  },
  Dojo: {
    name: 'Dojo',
    entities: {
      master: new Obj({
        name: "OldMan3",
        displayName: "Master Shi Fu",
        x: 8, y: 5,
        text: "Greetings young warrior, welcome to my DOJO! Where I teach people how to fight!"
      }),
      npc1: new Obj({
        name: "BlueSamurai",
        displayName: "Samurai",
        x: 2, y: 8,
        speed: 2,
        behaviourLoop: [
          { behaviour: "walking", direction: "left", tiles: 1 }
        ],
        text: "Training like this is the only way to keep my fists sharp and ready for anything."
      }),
      npc2: new Obj({
        name: "Master",
        x: 4, y: 5,
        behaviourLoop: [
          { behaviour: "walking", direction: "up", tiles: 2 },
          { behaviour: "walking", direction: "right", tiles: 1 },
          { behaviour: "walking", direction: "left", tiles: 1 },
          { behaviour: "walking", direction: "down", tiles: 2 }
        ],
        text: "I don't have time for idle chit chat, my training demands my full attention."
      }),
    }
  }
}

```

```

npc3: new Obj({
  name: "Monk",
  x: 6, y: 5,
  behaviourLoop: [
    { behaviour: "walking", direction: "left", tiles: 1 },
    { behaviour: "walking", direction: "right", tiles: 1 },
    { behaviour: "walking", direction: "up", tiles: 2 },
    { behaviour: "walking", direction: "down", tiles: 2 }
  ],
  text: "I must train harder to sharpen my skills and become one with the shadows."
}),
npc4: new Obj({
  name: "Monk2",
  displayName: "Tibetan Monk",
  x: 13, y: 6,
  behaviourLoop: [
    { behaviour: "walking", direction: "up", tiles: 1 },
    { behaviour: "standing", direction: "up", time: 5000 },
    { behaviour: "standing", direction: "down", time: 5000 },
    { behaviour: "standing", direction: "left", time: 5000 },
    { behaviour: "walking", direction: "down", tiles: 2 },
    { behaviour: "standing", direction: "down", time: 9000 },
    { behaviour: "standing", direction: "left", time: 5000 },
    { behaviour: "walking", direction: "up", tiles: 1 }
  ],
  text: "the path to inner peace begins with quieting the mind and finding stillness
within yourself."
}),
npc5: new Obj({
  name: "Lion",
  displayName: "Lion Warrior",
  x: 12, y: 2,
  speed: 2,
  behaviourLoop: [
    { behaviour: "walking", direction: "up", tiles: 1 }
  ],
  text: "ROAR! STOP DISTRACTING ME FROM MY TRAINING!"
}),
npc6: new Obj({
  name: "Villager4",
  x: 3, y: 2,
  text: "I'm struggling with these techniques, but I'm determined to improve and become a
skilled ninja."
}),
npc7: new Obj({
  name: "Monk",
  x: 11, y: 12,
  behaviourLoop: [
    { behaviour: "walking", direction: "down", tiles: 1 },
    { behaviour: "standing", direction: "down", time: 1000 },
    { behaviour: "walking", direction: "down", tiles: 1 }
  ]
})

```

```

        { behaviour: "standing", direction: "down", time: 1000 },
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 1000 },
        { behaviour: "walking", direction: "down", tiles: 1 },
        { behaviour: "standing", direction: "down", time: 5000 },
        { behaviour: "walking", direction: "up", tiles: 2 },
        { behaviour: "standing", direction: "up", time: 5000 },
        { behaviour: "walking", direction: "up", tiles: 2 },
        { behaviour: "standing", direction: "up", time: 5000 },
    ],
    text: "This is a sacred temple, a place of meditation and learning the arts of combat"
}),
npc8: new Obj({
    name: "Knight",
    displayName: "Knight",
    x: 14, y: 4,
    direction: "left",
    speed: 2,
    text: "I may be a knight, but sometimes it takes more than a sword to defeat your foes."
}),
},
exits: {
    map1: {
        name: "StartingTown",
        x: 8, y: 17,
        newX: 36, newY: 16
    },
}
},
FishingHut: {
    name: 'FishingHut',
    entities: {
        npc1: new Obj({
            name: "MaskFrog",
            displayName: "Frog Ninja",
            x: 6, y: 17,
        }),
        npc2: new Obj({
            name: "Boy",
            x: 4, y: 30,
            behaviourLoop: [
                { behaviour: "walking", direction: "up", tiles: 1 },
                { behaviour: "standing", direction: "up", time: 5000 },
                { behaviour: "standing", direction: "down", time: 5000 },
                { behaviour: "standing", direction: "left", time: 5000 },
                { behaviour: "walking", direction: "down", tiles: 2 },
                { behaviour: "standing", direction: "down", time: 9000 },
                { behaviour: "standing", direction: "left", time: 5000 },
                { behaviour: "walking", direction: "up", tiles: 1 },
            ],
        })
    }
}

```

```

text: "Welcome to our harbour, feel free to explore and enjoy the salty breeze!"

}),
npc3: new Obj({
  name: "OldMan",
  x: 11, y: 19,
  behaviourLoop: [
    { behaviour: "walking", direction: "up", tiles: 1 },
    { behaviour: "standing", direction: "up", time: 5000 },
    { behaviour: "standing", direction: "down", time: 5000 },
    { behaviour: "standing", direction: "left", time: 5000 },
    { behaviour: "walking", direction: "down", tiles: 1 },
    { behaviour: "standing", direction: "down", time: 9000 },
    { behaviour: "standing", direction: "left", time: 5000 },
  ],
}),
npc4: new Obj({
  name: "Villager2",
  x: 13, y: 26,
}),
npc5: new Obj({
  name: "Villager",
  x: 2, y: 23,
  behaviourLoop:[
    {type:"walking", direction:"up", tiles:1}
  ],
  text: "I may be small, but I work hard to earn my keep around here."
}),
npc6: new Obj({
  name: "Inspector",
  displayName: "???",
  x: 13, y: 18,
  direction: "right",
  text:"I'm sorry, I cannot disclose any information, I'm on a confidential mission."
}),
npc7: new Obj({
  name: "OldMan2",
  x: 8, y: 13,
  text: "Be careful not to step on the ropes there, mate, wouldn't want to send you for an unexpected swim in the harbour."
}),
npc8: new Obj({
  name: "Monk",
  x: 12, y: 12,
  direction: "up",
  text:"Hey there, want to help me load these crates onto the ship?"
}),
},

```

```

exits: {
  StartingTown: {
    name: "StartingTown",
    x: 3, y: 33,
    newX: 35, newY: 7
  }
}
}

window.mapDict = {
  StartingTown: new Map(window.Maps.StartingTown),
  Dojo: new Map(window.Maps.Dojo),
  StartingHouse: new Map(window.Maps.StartingHouse),
  FishingHut: new Map(window.Maps.FishingHut),
};

window.greetings=
["Hello there! Welcome to our town!",
 "Greetings, traveller! How may I assist you today?",
 "Hi! I hope you're having a great day so far.",
 "Welcome to our humble abode! How can we make your visit more enjoyable?",
 "Hey there! Is there anything I can help you with?",
 "Good day, adventurer! Welcome to our lovely town",
 "Greetings! It's always nice to meet new people. What brings you to our town?",
 "Salutations! I hope you're finding everything you need here.",
 "Hello, hello! I hope you're enjoying your stay in our lovely town.",
 "Hey, stranger! Anything I can do to make your visit more pleasant?",
 "Great weather, isn't it?",
];

```

Globals.js