COVER PAGE

Name – Bailie-Peirce Byrne

Student ID - 5328

Centre NO – 205750

WILD WASTELAND  
  
-A 2D MMORPG set in a post apocalyptic wasteland

Contents

[ANALYSIS 2](#_Toc435602651)

[Project Introduction: 2](#_Toc993852887)

[Features to include/Problems to solve: 2](#_Toc1483150422)

[Solution: 3](#_Toc511216161)

[Background Research: 4](#_Toc179853080)

[Similar Systems: 4](#_Toc53806105)

[Features from the systems: 6](#_Toc2079991837)

[Function research: 6](#_Toc2105845180)

[End-User Identification: 7](#_Toc201040346)

[Interview: 7](#_Toc1551442517)

[Written Analysis of response: 9](#_Toc1437898210)

[Existing System: 10](#_Toc1768293787)

[Objectives: 11](#_Toc1894219764)

[Modelling: 14](#_Toc1193224435)

[Prototype: 14](#_Toc1807889386)

[ERD FOR DATABASE 16](#_Toc1064241854)

[Class Diagrams: 17](#_Toc911840320)

[Other Modelling: 21](#_Toc938305286)

[DESIGN: 21](#_Toc2037959643)

[Design Introduction 23](#_Toc97776434)

[Data Structures: 26](#_Toc377270246)

[DATA STORAGE: 27](#_Toc1316642517)

[HCI (Controls/Interfacability): 28](#_Toc1171130097)

[Function List(Server Side): 29](#_Toc1736248729)

[Function List(Client Side): 37](#_Toc1057352008)

[Startup.py: 44](#_Toc1711792929)

[File Structure: 45](#_Toc1663302950)

[Key functions: 47](#_Toc850065041)

[Testing: 49](#_Toc772259043)

[TEST STRATEGY: 49](#_Toc1218267067)

[TEST PLAN: 50](#_Toc499610265)

[TEST VIDEO: 58](#_Toc837206842)

[EVALUATION 58](#_Toc678597557)

[Future Improvements: 59](#_Toc871989244)

[TECHNICAL SOLUTION: 61](#_Toc516786424)

[Complex Functions: 61](#_Toc1799914327)

[CODE: 62](#_Toc328584499)

[game.py: 62](#_Toc1164133360)

[SERVER.py: 133](#_Toc1825453773)

[Constants.py: 163](#_Toc1512475463)

[Items.py: 164](#_Toc1131907568)

[Startup.py: 166](#_Toc1989938215)

# ANALYSIS

## Project Introduction:

This project aims to be an MMORPG like the original Fallout series by Bethesda with multiple player instances existing in the world, within the world there will be NPCs, vendors, world bosses with server-side enemies and interactable objects such as doors and containers. There will be minigames within the world like lockpicking as well as projectile physics for weaponry with different calibre weapons being able to penetrate different materials and alter the damage done to enemies. Similarly, there will be a questing side to the game with multiple objectives and rewards such as new areas being unlocked.  
Within this project the player you control can move in a 2-dimensional plane with support for diagonal movement.

### Features to include/Problems to solve:

The main problem to solve is creating a multiplayer open world environment to be used as an MMO RPG style 2D game, this game should ascertain to the features from the fallout franchise while not directly mimicking the originals, the reason I have attempted to tackle this problem is recently developers of Triple A game studios have left 2 Dimensional game for unoptimized extremely resource intensive games that only a handful of end users are able to experience after spending sometimes thousands of pounds on a machine capable of running it. While I understand the progressive route utilizing newer engines, companies have been neglecting what pioneered games to their current state, for example popular titles such as (The Pokemon Series,Terraria,Fallout 1). My aim is to create a passion project that relives the simplicity of the older games while reviving the MMO RPG theme for my end users.

### Solution:

The way I will solve these problems is via the use of Py games drawing engine while also using TMX as a way to display different maps using tile sets, the player will be defined as a pygame sprite and contain an update function which will check for keypresses and alter where the player is drawn to give the effect of movement, an animation loop may also be included for a developed user experience, the use of tmx also allows meta data to be read by Pygame such as objects created on that laye of the map and convert those into objects accessible within the Pygame environment.

Movement can be handled via the use of a camera system in combination with co ordinated which will render different sections of the map dependant on the players location ensuring the player is always within the screen and draws a sizeable amount of the map to allow for exploration, movement will be handled by checking the button presses on the keyboard.

NPCS will be created via the use of Pygame sprites and will exist solely on the map that is defined to them in an external CSV file storing the NPC name and map thy spawn on as well as coo ordinates.

Items will be created using a csv file server-side holding all the identifiable values of the item as a dictionary a copy will also be made available client side to reference data pertaining to that item,(e.g image,stats). The items will be stored via a database and the only way to alter the contents is via the server-side to prevent, duplication mismatched item system

The inventory system will be created as a separate pygame loop displaying a window on top of the current screen and will draw, tabs containing the name of the inventory section pertaining to specified items, each tab will contain a list of items defined as pygame rects offset by an amount to give a list of items, which can be clicked and when selected will display the name stats and image of the item, of the item is able to be equipped then an equip button will be visible and once pressed will equip the item.

The server referenced will be a separate python file running server-side using sockets to handle connections, data will be transmitted via pre-defined packets which will be encoded using Json to convert them to byte array data and a ‘#‘ will be appended to the packet to act as footer for the current packet and header for the next so the server combine packet data and will handle each individual packet individually.

## Background Research:

### Similar Systems:

Similar systems compliant with the proposed solution must contain an open world environment with the presence of AI roaming the world and abide by the 2dimensional perspective of following the protagonist played by the end user. Similarly the systems much contain the ability to loot and store items found throughout the world with each item providing its own unique uses or benefits.

Firstly Fallout:  


REFERENCE ART FOR LOCAL ENVIRONMENT ^^  
  
The fallout franchise can be traced back to the 1990s when the parent company interplay entertainment released fallout 1 in 1997 a post-apocalyptic themed RPG, which is interesting as the classic fallout series are not multiplayer (Excluding the newer fallout 76). The game was handled by different studious branching their own spin-offs. However most, if not all fallout has a technologically advanced civilisation that spiralled into nuclear war with specific vaults holding survivors who are released after the bombs dropped and they begin to rebuild civilisation.

Fallout is widely praised for its innovative designs, storytelling and large open immersive worlds for the player to explore (Something I will try to Mirror). With human contact spread very thin it does create a sense of loneliness and gives the player a drive to explore and uncover the remaining secrets in the wasteland. Story telling is heavily prevalent throughout and is done via the variety of characters met while exploring. The older 2Dimensional games have a very simplified UI and focus on the occurrences of the map to tell story telling

Fallout is a dated franchise spanning back to the late 90’s it is an amalgamation and collection of various wastelands themed mmorpg story lines in which you follow to achieve the ending of a story or provide some closure to the initial opening.

Next is Stardew valley



Shown above is the 2-dimensional tiled game environment.

Stardew Valley incorporates robust RPG (Role-Playing Game) and resource management elements that add depth and complexity to the gameplay. In terms of RPG elements, players have the freedom to shape their character's identity and development. They can choose a profession that aligns with their preferred playstyle, whether it's focusing on farming, mining, combat, or crafting. This decision significantly influences the skills and abilities they acquire as they progress.

Resource management is another crucial aspect of the game. Players must carefully allocate their time, energy, and resources to maximize efficiency and productivity on their farm. Balancing daily tasks, like planting and harvesting crops, taking care of animals, and maintaining structures, requires thoughtful planning. Limited energy resources can be replenished by consuming food, further emphasizing the need to manage both time and inventory effectively. Players also need to decide how to invest their hard-earned money in upgrading tools, purchasing seeds, or expanding their farm.

Both the RPG and resource management elements in Stardew Valley contribute to a sense of progression, as players gradually become more skilled and prosperous while they tailor their experience to their own preferences. This rich blend of gameplay elements is a key factor in the game's enduring appeal and its ability to captivate a diverse audience.

### Features from the systems:

-Fallout AI

Firstly the AI from the fallout is a very simple and this a reason why it will be added to my project, a simple pathfinding algorithm is used to determine the position from the player to the enemy, dependant on the type of enemy, if the enemy is a melee enemy they will simply approach the player and follow an attack cycle inflicting damage on the player, this adds the need for considering what weapons to use and when its sensible to charge head first into groups of enemies. The next is ranged enemies dependant on the type of weapon they are using they will approach the player until the weapon they have is effective at the range they are currently at and will open fire on the player inflicting damage, once again providing an extra layer of strategy to the game, in which prioritizing the most damage inflicting enemy is a sensible choice especially those at range which are harder to target when swamped by other enemies.

-Fallout Inventory

The inventory system from the fallout franchise has been quite simple and user friendly to the newer end users providing an efficient and useful UI to manage and control items obtained and allow transfer between containers. The transfer window also like the inventory overlays the current screen with two distinct windows the players and the accessed containers. These windows are very self-explanatory and allow the visualization of moving items between containers, furthermore the only information displayed is the Name, Image, and Stats of the item. The reason to include this is purely for functionality combined with ease of user access and understanding it is an abstracted version of resource handling utilizing information hiding to only display relevant information for the scenario.

-Stardew Valley Map

Stardew Valley employs the TMX (Tile Map XML) format for its map creation, which is a widely used and versatile file format for defining 2D tile-based maps in games. The use of TMX in the game allows for several key advantages in map design and customization.

Modularity: TMX allows the game's developers, to design and create individual tiles and tilesets that can be easily integrated into the game. This modularity simplifies the process of adding new content, whether it's new areas, buildings, or terrain elements, without having to overhaul the entire game.

Efficiency: TMX facilitates efficient rendering of maps by breaking them down into a grid of smaller, reusable tiles. This approach reduces the computational resources required for rendering, improving performance, and enabling the game to run smoothly even on a variety of hardware configurations.

Customization: TMX provides a high level of flexibility in map design. Tilesets can be customized to create unique environments, and layers can be added to handle various aspects of the game world, from terrain and buildings to weather effects and dynamic elements like character paths and events.

### Function research:

A\* Pathfinding:

A process I have researched is arguably one of the most crucial features being pathfinding allowing the movement of non-playable characters and enemies alike. A\* is a Dijkstra algorithm used to find the shortest path from one node to another while using a heuristic to further improve its efficiency. Firstly, this algorithm favours vertices close to the beginning node and information that a similar greedy best first search used which favours vertices closer to the goal the algorithm then deduces a cost being the amount of nodes needed to be travelled through to reach the goal. The reason the A\* pathfinder will be used is due to its grid Matrice applications. Such applications allow me to use TMX tiles as nodes and henceforth path find directly using a map I provide, as well as the ability to specify specific node values as un-walkable. A\* was first developed in 1968 to combine heuristic approaches with formal approaches, combining the approximation from the heuristic with a thorough formal approach provides the shortest path. The big O time of the A star algorithm is substantially less than Dijkstra's algorithm and similar algorithms.

Larger companies such as google use A\* for projects like Google Maps as well as standalone apps such as Waze, due to its speed and flexibility in navigating nodes at more efficient speeds and only seeks the most efficient route.

## End-User Identification:

The system I am creating is designed for a wide variety of gamers, and those who enjoy video games and the ability to explore an open world environment, this system also suits RPG fans who enjoy immersing themselves in a situation or world created.

### Interview:

The candidate I have chosen to interview is Thomas Clayden a gamer and developer of his own video game.

|  |  |  |
| --- | --- | --- |
| Question | Response | Additional |
| Is the system easy to use and understandable by yourself? | Yes the game is very easy to understand and get into as it works and plays very fluidly, as a gamer it works the same as most other games meaning that there wasn’t much confusion over any of the features and the system | A New menu has been added to act as a tutorial, as well as provide explicit information regarding the use of the system. |
| Do you find the system immersive as a gamer yourself? | Yes the mechanics of the game work very well with each over and are definitely fluent | New soundtracks and Points of interest are planned to be added to increase immersion. |
| Are the mechanics of the game such as movement and gun play fluent? | Yes the mechanics of the game work very well with each over and are definitely fluent | Updated enemy movement and polished gunplay to ensure smoothness. |
| What features did you find the most difficult to use? | There wasn’t any feature that was difficult to use all of the ways that the game worked made sense and worked very similarly to most other games so there wasn’t any confusion for me | Once again an updated UI has been added to further decrease the difficulty in understanding any of the games features. |
| Which Feature was the easiest to use and why | The easiest function to use was the killing of enemies as it as simple as selecting what weapon and then moving to the enemy and then clicking it worked like every other game so it was very easy to use and functioned very well | Plans to update enemy AI to have multiple states such as running away as well as adding new types that use explosives or weaponry. The goal is not to make killing enemies the easiest feature as that is the majority of the difficulty in this system. |
| What do you believe the purpose of the system is? | The purpose of the game is to be a multiplayer game quite similar to fallout where the user has to kill and survive in a radioactive waste land and this does a very good job of doing so | He nailed the purpose of the system, and even got the correct game from that provided inspiration. More assets are planned to be added from fallout and similar games. |
| Would you use this system yourself if so why, if not why? | Yes I would use this system as the way that the multiplayer and enemies work is very good as it all runs very smoothly and the path finding is utilised very well in companionship with interpolation | N/A |
| Are you able to run this system on your own devices? | Yes | Credits the compatibility and ensures there is no memory leaks as well as the CPU time not being demanding. |
| What feature would you like to see added? | N/A | N/A |
| Were any bugs encountered when playing? | There was only a slight bug that I found was when moving around the map and going down so that the camera started moving the player health wouldn’t stay above them so when you hover above them it wont come up this hit box would be else where | This bug has now been patched and was accredited to not accounting for the camera offset when moving across the map, this has since been fixed.  Camera offset has also been taken into account for other functions such as projectile motion e.t.c. |
| How was the difficulty in terms of weapons accessible alongside enemies? | There wasn’t much difficulty for either as the user interface provided plenty of help and explain when I didn’t have any ammo to stop any confusion | UI seems to be explanative enough. Gun play remains a polished aspect of this system. |
| How do you feel about the graphics and audio? | The art style of the code is very good I like the way the tile set looks and the way you have set out the map. As well the adding of audio is very nice as it makes the game feel less empty and boring and adds to the instructiveness of it | More Music is planned and the use of PiSonic to create SFX for when map events occur. |
| Any parts of the system you felt boring and or repetitive? | No | New NPCs have been added and storyline has been expanded to increase content and decrease end users boring. |
| Would you replay this game to explore with new items or new areas? | Yes that was the only bit of the game I would recommend adding would be new areas and maps so if it was added I would be very inclined to replay it | New buildings and maps are planned to be created.  Procedural Vaults are also planned. |
| Out of 10 what would you rate the game according to your tastes and why? | 9/10 as it is a very good games and works very well and as intend it is very easy to play and to understand while also being enjoyable at the same time with only minor bugs and thing to add to it | Majority of bugs have now been patched, additional testing is required to confirm.  TMX art style is also planned to be expanded on. |
| Finally, would you recommend this game to others? | Yes I would definitely recommend this game to other | N/A |

**Additional Response:**

The system was very easy and teaching to the user as when going around through the map anything you could interact with there will be a piece of user interface to tell you so and in a lot of the cases tell you what buttons to press such as going in a building, opening chests and talking to the NPC as well this user interface is very nice as it gives the user a lot of information such as if you have ran out of ammo it will tell you as well as what armour and weapon you are currently using and the damage reduction it give to you making the game very easy to understand and traverse around the map.

**Does It Meet The Objectives?:**

The code after playing it I do think that it has met all of the objectives that was set and that they all work tell and as they were intended to.

**Any Criticism?:**

Overall there isn’t a lot or much criticism that I could come up with the code as it works very well and all runs very well and user friendly the only criticism I would say would be the smoothness of the enemies movement as I can see from the code interpolation is already used to smooth out the enemies movement greatly but if possible to improve it that would be the only area that I have criticism

**Improvements or Extensions?:**

There isn’t really any improvement to the code as it stands apart from what I already mentioned. Then the possible extension I would give for this system is adding new things such as adding new maps and worlds for the user to explore as of currently there is only the one world and nothing much apart from the main area, but in that main area there is a lot to do and to test and explore and this isn’t a full game so if it was to be carried on the next extension would be to increase the worlds and things to do around them. Another extension I would give would be to add some more animations to the game that make it look a little cleaner such as multiplayer adding animation for the other player as well as when they attack so you can see each other attacking and doing certain tasks.

### Written Analysis of response:

Upon review of the response provided by my selected candidate it is made apparent the game is user accessible and the user understood how to interact with the game world and utilize the weapons and items as intended. Future improvements suggested by said user is the polishing in terms of enemy movement, which is an error I have identified server-side it seems the queue structure is popping the path the enemy needs to take too early causing the interpolation client side to be off. This leads to the aforementioned jittering and rough nature of the enemy’s movement. The current main point of criticism I have identified in the response is the lack of content, as expected for a prototype the map and current enemies are sparse leading to the user to extinguish the majority of the content rather quickly.

However due to the nature in which I have programmed this project the ability to create a new enemy is completed with a single line of code, likewise with the maps it’s a matter of importing a tile set and drag and dropping these tiles into a free 3rd party compiler called tiled and exporting the resulting CSV files to the projects specified directory.

Overall I will try to implement a sure set “win” feature to allow the player to strive for and keep the game fresh and entertaining for the user, more buildings and NPCS are planned for development alongside quests.

Upon thorough review of the response, I am pleased with the users satisfaction and kind words about the system and its functionality and have seen it has been a positively received idea, allowing me to work confidently while listening to processing and utilizing the feedback provided.

## Existing System:

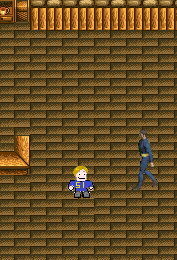
The system is a game containing various enemies, buildings and NPCs the game is played by controlling a character across a deserted wasteland scavenging containers and completing quests to gather enough currency to be exchanged for your freedom from this wasteland to a promise land sanctuary. The wasteland will have to be traversed to locate the place to exchange this currency.



Attached is the overworld map visible is the objects found within the layer all objects are coded to have collision with the player rect hitbox giving an immersive polished experience. The objects visible can have properties assigned for example, I can assign the interactable property to the object allowing the player to interact with them. Furthermore, assigning specific IDS to containers or values dependant on the material of the item for bullets to be affected.

The game is designed to be played single player however multiplayer exists as a way to play with friends while helping one and other. NPCs can be found around the map and interacted with by pressing key binds these will give you quests to complete for items and currency.

Essentially the player must navigate a hostile open world environment while liaising with characters to upgrade their gear and face stronger and more aggressive threats, all while continuing to journey towards this promise land.



Ctypes is a preferred library to obtain screen dimensions and system allowing universal Fullscreen independent of system prerequisites once again providing quality of life improvements for the end user once again contributing to a smooth enjoyable experience.

Math is an import used to handle more advanced calculations prevalent in the code such as the use of Trigonometric functions found specifically in the lockpicking game to determine the angle the users lockpick is compared to the true north line of the game as well as the math library contributing to the interpolation of enemy movement client side to provide a more synchronized enjoyable player experience.

Imports is a directory I created to store required files like the constants and items list providing the needed info to the client for functionality and compatibility for the server.

The game is simple and is a very barebones framework of an mmorpg with the ability to move in the 4 cardinal directions the main purpose or objective is to locate NPCs and complete tasks for them in order to further story progression as some actions are gated behind quest requirements, the story however is but a small section of the game as the world being open environment pertains to the fallout open-world adventure side. Where fallout is not usually multiplayer, I have adapted this game to incorporate multiplayer elements and allow interaction between clients via the sharing of damage between enemies as well as the ability to share items and loot using containers found scattered along the wasteland.

There a two main types of interactable present, one being doors the other being containers as aforementioned containers can be accessed to retrieve and or store items as the loot is procedurally generated at intervals set by the server.

Doors on the other hand are specifically used to traverse between different maps allowing one player to move to another map, only visible to players also on this map. Each map has its own independent enemies as well as containers and scenery to be explored and looted.

## Objectives:

* Enable a 2D sprite to move in the 4 cardinal directions using pygame sprites
* The Pygame sprite should be able to be controlled via the use of either the arrow keys or WASD to comply with the most universal standard of movement in video games.
* An Animation Loop may be added for further immersion for the player, this will be done using the modulo function to loop through an existing list of animation frames.
* Draw a map using a tileset for the player to move on
* Pygame should be used to read the provided TMX file by iterating through each layer readin the provided values found in the CSV file and apply these values to that of each individual tile and display them accordingly to the map surface defined
* The player should be drawn on top of this layer and traverse it.
* The map created in tiled should be exported alongside the CSVS to be used as pathfinding layers.
* Allow that map to be changed via interacting with objects like doors
* Different maps should be loaded when switching between layers whether that be via the use of door or dying and respawning back at the start of the game.
* There should be a notification when interacting with the objects to change maps allowing a heads up to the player.
* Each map should be sufficiently detailed to provide an immersive experience and each map should contain its own enemies, NPCS and interactable unique to that map
* Create collisions within the server objects to create Boundaries for the player
* When switching map a function shall be called to gather the colidable objects such as walls houses, containers e.t.c
* These objects should be converted from the TMX rectangle object to a pygame rect and stored withing the current Map Instance object list so when the player moves it can compare using the colliderect() fucntion
* The player should easily be able to tell if an object is passable or not for example the objects must not be small enough that the player doesn’t know why they are unable to move.
* Use Sockets to form a client server connection and allow bi-directional transmission of packets
* The python library sockets will be used firstly when the server becomes active the s.accpe() function will act as a stopper so a thread will have to be used to handle connections. When a connection is established the clients socket will have to be stored to later be referenced when sending packets to said client.
* Packets will be predefined upon launching the code to prevent having to define a command everytime it is needed
* Packets should contain either a footer or header being a wildcard character to allow the packets to be sliced at the correct point to isolate the data between each packet and prevent collisions
* Create enemy instances that use pathfinding and traverse the map to attack the player
* The Enemy instance should be constructed by passing data from a CSV file which will contain the stats and data in regards to that enemy, to a class method which will construct that instance using the value sfrom the CSV file.
* The enemy class will be added to a Pygame sprite group when created for refeferncing at a lter time as well as keeping all instances in one place to ensure no instances are lost causing a memory leak.
* Each enemy should have a method to calculate the distance between them and the nearest player if there is one, doing so it will compare the nearest to its range, a predetermined stat, and if the player is within that range it will pass it to the pathfinding function.
* The enemys pathfinding function should take the target players position and its own position then using the A\* algorithm will calculate a list of tuple coordinates being the path and add them to its self.path list which will be passed to the move function.
* The movefunction will take the first element of self.path convert the grid co ordinates into world environment co ordinated by multiplying by 32 to match the grids, the enemys x and y will then be checked and if its less it will be incremented until it reaches the players co ordinates giving the enemy the ability to move.
* After each movement the player should receive the enemys current tile so the client can accurately display the movement of that enemy and use interpolation for a smooth movement.
* Use SQL database structures to store player positions and other data like container inventories
* A table named containers will be used having an ID for each container as the primary key, then contains every column of item in the game e.g(Weapons,Armor…) in these columns a json dictionary will be stored containing the items and their amounts.
* Each container should also have a locked Column to indicated whether or not the container is locked and requires a lockpick to be accessed.
* Add Multiplayer so multiple instances can be connected to the server and their positions be synchronized unanimously across all connected clients.
* When two clients are connected and on the same map layer a sprite representing them should be visible to the other connected players
* When a player moves the sprite representing them on other clients should also move to mimic your movement and effectively display multiplayer
* The co ordinates stored in the database should be used so all players see the same other players in the same sport providing synchronization.
* Add NPCs that have interactable dialogues and provide quests for you to follow and complete
* NPCS should be map specific and will be represented by a sprite
* NPCS spawn location will be stored in a CSV file and similar to the enemies the data regarding the name of the NPC the co ordinated and image will be fetched and passed to a class method to construct that instance.
* Each npc will have their own dialogue dependent on the players quest completion as well as items in their inventory and of course what the name of the NPC is.
* Add interactable Containers with their inventory's accessible via everyone
* Containers exist server-side meaning their contents are universal for all players and accessible by all
* They work by first colliding then interacting with the Containers interactable Pygame Rect, once this has been achieved the client will check if the container is locked if not will request the contents from the server.
* If the container is locked and a lockpick is present within your inventory, then you will be prompted with a lockpick mini game in which if you’re successful will unlock the container for every player as well as yourself.
* Create an item storage system and allow the use and collection of items.
* Items can be viewed within the players inventory clicking on the tab that corresponds with the item type will display that items information.
* If the item is a weapon or Armor, then the equip button will be available and when pressed you will equip said item.
* Consumable items can be equipped into the offhand and when a key bind is pressed will be used and an effect will be applied to the player.
* When firing a weapon a projectile is made
* Each different weapon uses different calibers to fire and said caliber has its own unique effects
* Ammo should have varying speeds and penetration ability
* When the player clicks a bullet will be created using the co ordinates of the mouse and source and will then move towards the target at the provided speed
* The bullet should also ensure that if it leaves the game world that it is destroyed to prevent a memory leak.
* The bullet should loose speed as it travels.
* Allow the items to have individual stats and properties unique to one and other
* Items which are defined in the items.py dictionaries exist as a tuple of themselves as well as the data pertaining to that item.
* Each item has their own set of values, if this is armor the value will be the % reduction of damage take from enemies
* Weapons will have separate fire rates, bullet speed, and damage
* Add map dependancy for enemy tracking so if a player switches map the enemy stops tracking them
* Every players position and map will be stored in a dictionary server side which will be constantly used to update the enemies dictionary
* The enemies should cross check their own MapID with the players if it does not match then they will be ignored as a possible target, however if they are on the same map and are in range then the enemy will track them.
* Add UI and HUD elements for easy viewing and uncomplicated instructions.
* Equipped items should be displayed on the screen.
* The stats and benefits the equipped items have should also be displayed.
* Notifications should appear when a world event happens such as completing a quest.
* When the ability to interact with an object exists then a message should appear indicating so.
* Utilize threading for autosave features client and server side
* Client Side an autosave thread should exist this will take the current position of the player, the map and the current health of the player will be sent via a packet to the server every 0.3 seconds. The server will then update the database with this data ensuring it is saved.
* Add exception handling to server disconnects or database errors.
* There should be no Unreachable code within my system.
* Larger functions should be wrapped withing a Try Except loop to ensure any errors are caught and displayed to the console for server management to view.
* Plans to add a server.log so any exceptions that occur are saved for debugging issues.
* Utilize external files like csv, py, tmx for variables and constants.
* External directories should house all the assets for the game such as PNGs as well as background music
* Mundane and frequently used constants will be stored in an external python file and referenced when needed upon running.
* Pathing CSV and tsx files will be stored within the maps directory for access when constructing and loading the map and its collisions.
* Splintering and sorting external files into group directories will greatly help the accessibility of needed files while runtime is active.
* Menu to display keybinds/instructions
* A menu should be available at any time to the player displaying instructions on how to play the game.
* The Menu Should be straightforward and as simple as possible
* It should Contain all the possible keybinds to ensure full compatibility
* New players will have this menu force opened upon joining.
* Add serverside container saturation
* A server side script should be ran upon launching the server which will iterate through a list of the items that exist in the game.
* The script should then generate a random quantity using the template in combination with the container ID to add an element of scaling as containers with higher IDS will be found later in the game world.
* The script will access each container decode the json strings containing the items for each category, then generate new items if they exist already in the container they will be incremented by the amount generated. If they do not exist then they will be added before being converted back into text and stored in the database for use by the players.
* Items should be able to be blacklisted and not occur naturally in containers.
* Add server-side rewards for killing enemies
* The server will receive packets from clients indicating when that client damaged an enemy.
* Once the health drops to 0 or less the server will award the player a random amount of currency
* Items are planned to be dropped by enemies too.
* Sort quests using sets to ensure only one quest of that type can be active
* Due to the nature of a quest only one of that quest type should be active at once for example the player shouldn’t be able to have two instances of the same quest. This can be achieved by indexing the quests using ID and using a set to store them client side. Likewise, server side there will be a check to check each players individual quests and the ones that don’t have that quest will be given the quest at completion stage 0

## Modelling:

### Prototype:

Below are some snippets of the prototyping of different elements working in synchronization to provide a working functioning system as aforementioned.

Socket connection is used to handle the Client Server relation by sending Json encoded byte strings to each client that connects to the server. All socket connections are automatic and connect to the given IP and Port provided in the Constants.py File. Likewise, the client can receive packets of data correlating to other in game references and build a class based on them and deploy the instance within the game world. All incoming packets are assigned a header in this case ‘#’ to ensure minimal packet loss and concise packets.

def recv\_server():  
 global pmap  
 global xpos  
 global ypos  
 global hp  
 global map  
 global enemies

while True:  
 data = s.recv(2048)  
 try:  
 print(data)  
 data\_list = data.decode().split("#")  
 for msg\_str in data\_list:  
 if msg\_str != "":  
 msg = json.loads(msg\_str)  
 if msg["COMMAND"] == "INIT":  
 xpos = msg["X"]  
 ypos = msg["Y"]  
 hp = msg["HP"]  
 pmap = msg["map"]

Above is a snippet of the client's code responsible for receiving decoding and executing commands from packets received from the server. As mentioned, the Json byte array is decoded and serialized within the message string before its contents are then checked and executed if able. All packets are formatted to contain a dictionary with the COMMAND header this is so whether the packet be arriving at the client or at the server the code is understood in terms of the data contained.

===========================================================================

Threading is also prevalent within the game, firstly the thread allowing sockets to run simultaneously to the main game loop as well as a few other threads which handle things like saving data, updating position, opening containers and minigames found within. The use of threading sits at the forefront of the code's compatibility allowing tasks to execute smoothly providing the end user a smooth and enjoyable player experience. Threads are managed very precariously minimizing cpu time. A single thread is created when a new client joins the server and is closed when they leave. A single thread also manages the enemy AI and movement only handlign their movement if conditional statements are met to futher minimize CPU times.



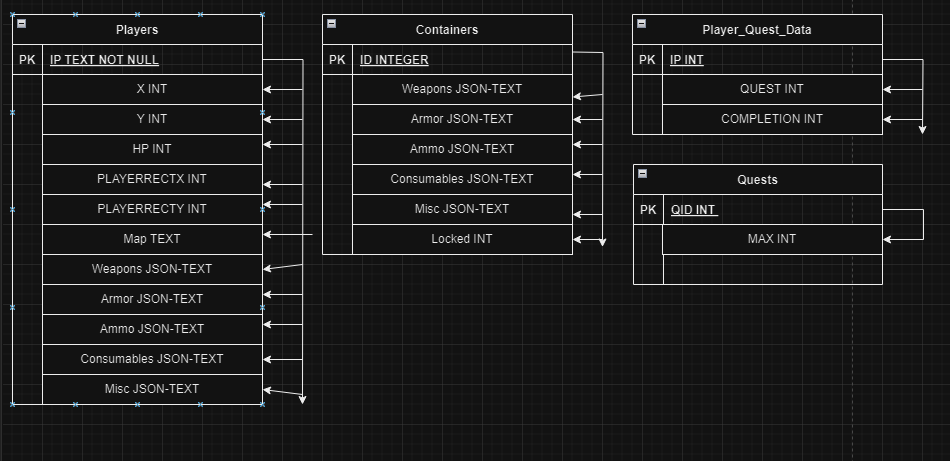
Above is the use of threads demonstrated by enabling the recv\_server function which is recursive but contains the stopper command data.recv() meaning it will only run when data is received from the server, however, will run consistently in the background allowing the user to interact with the client and Gameworld independently. Due to the project being a client-server model I must ensure compatibility between the two in order for the system to work effectivley.

Shown below is proof of the current display of the prototype containing a single enemy and the player instance.



### ERD FOR DATABASE

DATABASE.DB:



**One-to-Many Relationships:**

Players to Player Inventory Categories: There is a one-to-many relationship between the "Players" table and the player's inventory categories (e.g., Weapons, Armor, Ammo, etc.). Each player in the "Players" table can have multiple items within each inventory category.

Containers to Container Contents: There is a one-to-many relationship between the "Containers" table and the container contents. Each container in the "Containers" table can have multiple items within each item category (e.g., Weapons, Armor, Ammo, etc.).

Players to Player Quest Data: There is a one-to-many relationship between players and their quest data in the "Player\_Quest\_Data" table. Each player can have multiple quest entries based on different quests that they are currently have active.

**One-to-One Relationships:**

Players to IP: Each player in the "Players" table is uniquely identified by their IP address ("IP"). This

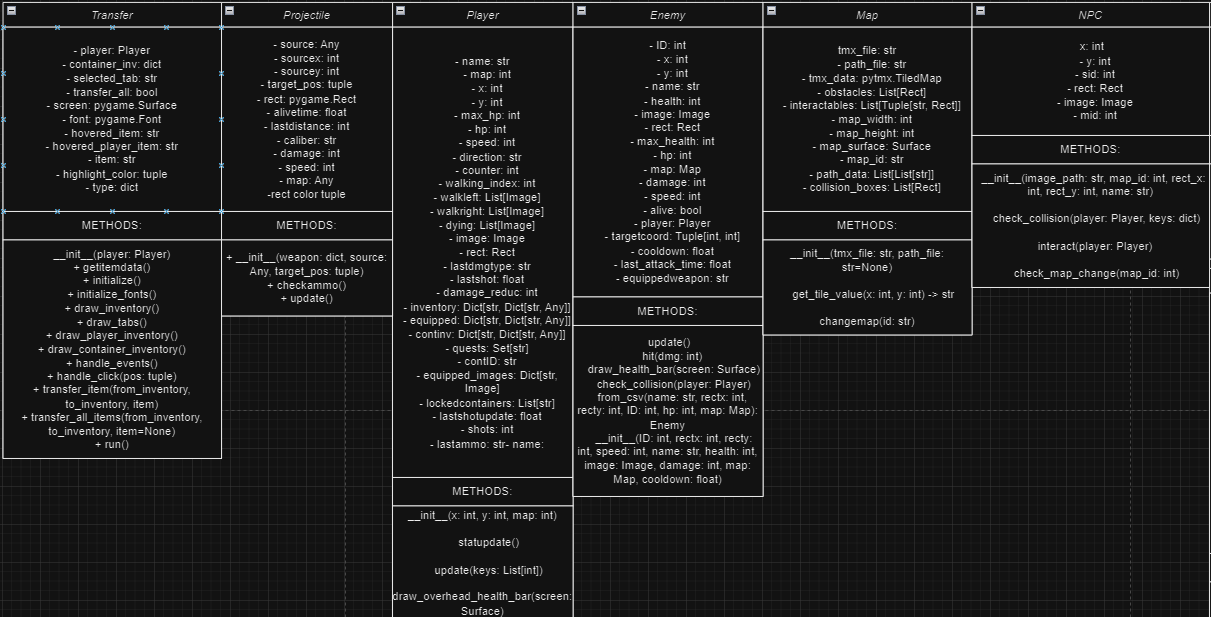
Containers to Container ID: Each container in the "Containers" table is uniquely identified by its "ID."

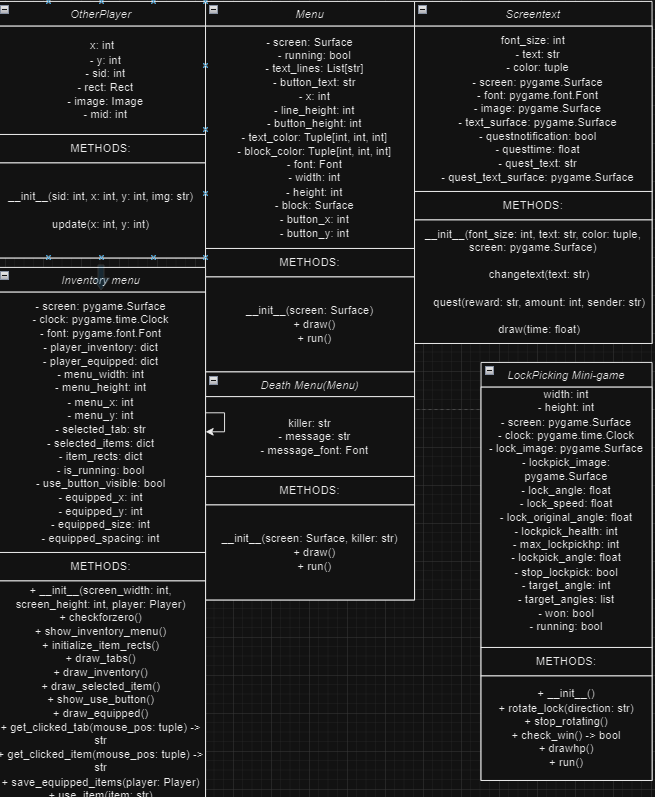
Containers to Locked Status: There is a one-to-one relationship between the "Containers" table and the "Locked" status for each container. Each container can be either locked (1) or unlocked (0), and this relationship is conditionally determined.

Player and Quest Data: The quest\_update function updates a single row in the "Player\_Quest\_Data" table. It modifies the quest stage (COMPLETION) for a specific quest (QUEST) for a particular player identified by their IP address (IP). This represents a one-to-one relationship between the player's IP, a specific quest, and its completion stage within the table.

## Class Diagrams:

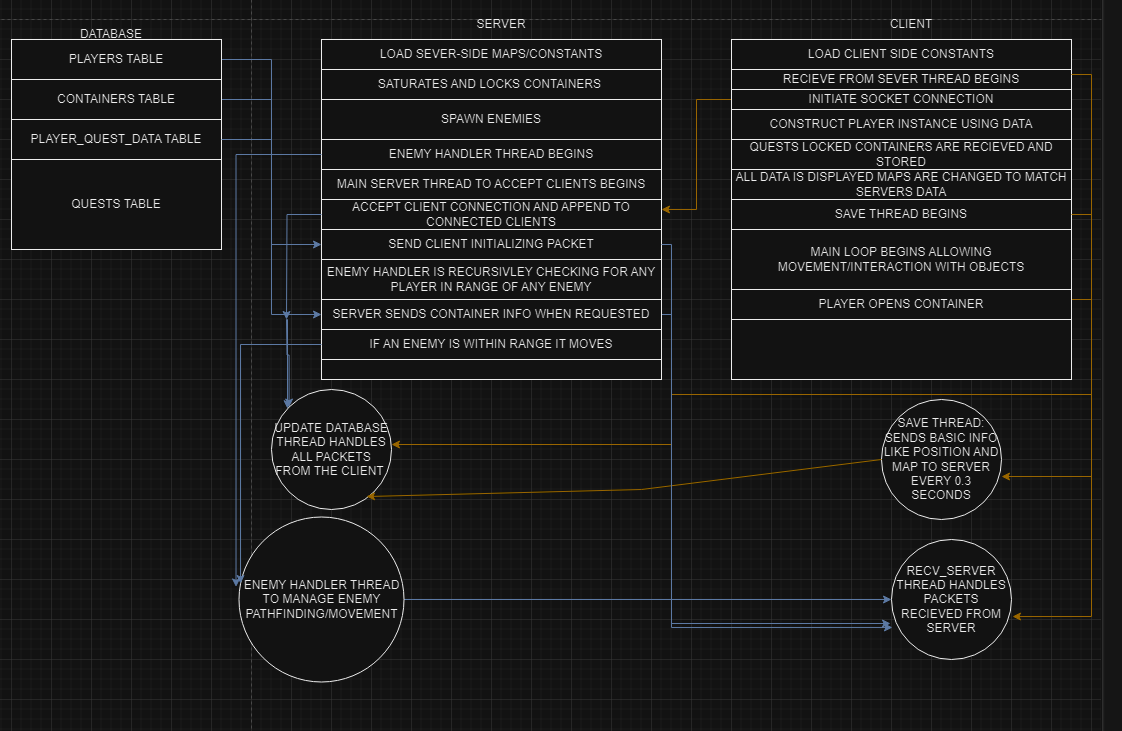
Provided is the client side gaem class diagram containing all existing classes and their corresponding varables as well as the data type corresponding as well as all methods for those classes.



Server side only one class exists and that is an almost carbon copy of the enemy class, this is because the server uses the players position recorded by the database and simulates the pathfinding and movement server side before passing the updated enemy's position to the clients, this is to ensure all clients are viewing the same enemy instance at the same position. This is to place less strain on the client to allow lower end hardware to play this game also, increasing the accesiblity and thus size of my intended users.

|  |
| --- |
| Enemy |
| - id: int  - name: str  - health: int  - speed: int  - damage: int  - path: list  - x: int  - y: int  - map: str  - mapdict: dict  - rect: pygame.Rect  - dict: dict  - lastupd: float  - alive: bool  - grid: Grid  - range: int  - fleeing: int |
| METHODS: |
| + \_\_init\_\_(name: str, health: int, speed: int, damage: int, x: int, y: int, map: str, id: int, range: int)  + check\_nearest()  + move()  + from\_csv(name: str, x: int, y: int, map: str) -> Enemy |

### Other Modelling:



# DESIGN:

My system is a multiplayer networked client-server model designed as a 2-dimensional multiplayer RPG game framework in which the client uses sockets in a client server relation to effectively and candidly supply information to and from the central database. This database contains information such as player inventories and quest completion. The system is designed for gamers to experience a multiplayer experience with multiple enemies and interactable as well as a questing system.

The project will be split into 3 major areas, the server, the client and the database.

Additional files will be used containing Assets such as PNG files, or map TMX, TSX, CSV data. As well as Constants.py and separate minigames named accordingly.

Both the server and game are python files utilizing pygame as their logic engines as well as tmx to handle the displaying of the Matrix based map. A matrix was chosen at it was efficient to include the pathfinding algorithm and incorporate that into the enemy's movement, alongside the swift ability to create new areas of the map as well as create new maps entirely simply utilizing tile PNG images.

Server-side once the player connects to the server the database will be checked for any existing data pertaining to that player via their IP as the key. If data is found, then an INIT standing for initialization packet is sent to that client containing their data from the database for the client to load and store. If the client does not exist then a template default INIT packet is sent containing the data of a new player, the server will then append this client's data to the database for later storage. Once the client has connected and all relevant data pertaining to them has been sent the client can begin interacting with the game world. Every 1/3rd of a second the client sends the server a packet containing their basic information such as location,map,health but not inventory, this is to keep the packets to and from the server as small as possible to prevent lag or overloading my server. The server will amend the database with the new data essentially acting as an Auto-Save feature. The client can access containers doing so will first check if they are locked, if containers are locked the server send that information alongside the INIT packet, if the container is unlocked the client sends a GET request for the contents of that container, the server will then return that data. Any items will be displayed via the client-side transfer window allowing them to transfer items between their inventory and the container. Each action is also sent to the server and the server handles this by moving and or updating the players and containers items. This server-side sanitisation will ensure no items are duplicated or that client-server data mismatches occur.

If said container is locked the clients inventory will be checked for a lockpick, if one exists then it will be removed and the lockpicking window will be opened, this minigame allows the user to attempt to unlock the container, doing so will continue the clients GET request for the contents , while simultaneously updating that containers locked status server side and unlocks the container for all current connected clients also.

The client is will be able to equip Armor and weapons found in these containers or from NPCS or killing enemies, doing so will be available from the inventory menu. The inventory menu will follow that of Fallouts in terms of items being split into categories with each tab containing the items belonging to that category, each individual item will be stored alongside its quantity. When pressed the item wills information will be displayed and if the item is equippable the Use button will be shown. Different items have different effects such as damage reduction or healing.

Next will be the enemies which the server thread will iterate through their sprite group and check any enemies which are on the same map layer as the client if so it will check the distance between them and if the player is withing their range the enemy will begin pathfinding and moving towards the player.

## Design Introduction

|  |  |  |
| --- | --- | --- |
| MAP LAYER | The designed aim is to have a map blit to the screen using pygames screen. Blit and .update function to display a custom created map to the users display. The map should eb defined as a class with the Obstacle attribute which stores the individual hitboxes for objects such as walls or inaccessible areas. The map layer should have a function which allows the map to be changed and the user position be updated alongside the obstacles for that intended “layer.” | The Map will be split into 3 files, the TMX, the CSV and the TSX files. The tmx will be used to handle Object identification and classification, the CSV will be used as a Matrix for pathfinding and the TSX file stores the individual assets to be mapped to the CSV file to display the maps tileset. |
| PLAYER INSTANCE | The player instance is the basis to define the users actions and is visual output according to their action. The player similarly to the map is defined as a class with its own methods such as update which is a constant check to record user input like keys and move the player on top of the map accordingly. The player instance also handles collisions verifying the desired place to move to is not within an object. The player instance stores its own player stats such as health, speed and all other S.P.E.C.I.A.L stats. The player will also have an animation loop depending on the actions. | Primarly the Player (A pygame sprite) is able to move in the 4 cardinal directions (N,E,S,W), interact with map Objects like Containers, or Fire their Weapons at Enemies. The player sprite is visible to other connected players. |
| OTHERPLAYER INSTANCE | Other player instances are simply pygame sprites that contain the image they appear as and a unique sprite ID. These are used when another player joins the server, and the other player appears as this sprite. These sprites are accurate to the other players current position to give the multiplayer element. The sprite should not be interactable with the player instance or any client-side instances. However, they should be able to communicate using the chat function and trade | The other player instance is what is used client side to represent other players positions orientation and actions. They are essentially the other players existing within your client and are displayed by their own pygame sprite individual to whatever map layer they exist on. |
| CAMERA SYSTEM | The camera system handles what part of the current map is displayed to the screen surface layer. First the program updates the camera to ensure the player is on the screen and not allow scrolling off the map using the map size. Then render is called which prints the player on the screen surface on top of the map layer, | The camera system defined as a class handles the movement of the camera as well as other GUI elements such as on-screen text and is responsible for handling the focusing of the player on the screen. |
| Enemy Instance | The enemy instances are pygame sprites which as expected act as enemies that, exist both client side and server side, all pathfinding is done server side then the path information is sent to the client to handle movement, ensuring no collision with objects. The enemies have their own attributes such as health speed damage. | The enemies are split between Ranged and Melee type enemies which essentially dictate their distance before engaging the player. Some enemies may carry weapons, guns and have the ability to throw grenades much like the player. |
| Projectiles | Projectiles are used for bullets, grenades, rockets. Currently they are client side but are planned to be server-side. Projectiles can be hostile or allied in which they will either do damage to enemy instances or player instances respectively. | Projectiles will interact with the objects found within map TMX files to handle Physics such as penetration like materials or would be susceptible to lower calibre weaponry whereas metal/concrete will be penetrable by higher calibre rounds. |
| Interactables | Interactable are a broad range of objects which are interactable with such as Doors, Containers, Traps, Levers. The interactable class is home for all inventory transfers between containers. The items that exist within the game exist in a Closed Loop meaning items cannot leave nor enter the system to prevent Dupe glitches | Interactable are server side objects meaning their contents are communicated to and from the server to establish a global unison between all containers, and game states of objects like levers or Locks. |
| NPCS | Non playable characters, will be passive entities existing in a fixed space within their map Layer and can be interacted with to receive Quests, or some NPCS may offer specific deals, or some may be able to barter with to provide new items. These NPCs will be client side yet the Quest Data they provide will be dependent on server-side data such as completion status of other quests or Progression Gating like Level/Skill Requirements. | These NPCs will be vital to the game to allow progression and give a more polished feel to the Intended Target Audience as well as add to the immersion. They also act as a vessel for storytelling and providing a platform for shops. |
| Questing System | The questing system will be the way that story-telling and immersion is handled in the system with quest data being stored server-side and relayed back to the client when needed, completing different quests will provide different unique rewards | So far only 2 NPCS exist as a proof of concept, but this willlater be expaned with each NPC’s Interaction varying on whichever stage of a quest you are at. |

A diagram of a company

Description automatically generated

## Data Structures:

Firstly a collection of dictionaries are used to store the data of packets found at Lines 29-36, these contain the specific commands and values recognizable by the server. All client side packets are in the format {COMMAND:X,VALUE1:X…}where the command contains a key word recognized by the server, these packets are serialized using json formatting into byte array data send via sockets and decoded server side and then executed.

A map dictionary is used to store a key and tuple containing firstly the map ID as its key then the tuple value contains the TMX file and the CSV file used by the client to display the correct map and pathing data.

A list will be used to keep a track of sprite IDS on the server when an enemy is created that sprite is assigned a UUID and when the server sends clients the other player data its universally recognized and data regarding that player is shared all clients simultaneously. This sprite ID also acts as a key to reference players and ensure they only appear on their corresponding maps. This is the same for the Enemies list and has the same usage scenario client side.

Pygame Sprite Groups are used to store and index the values of instances of classes that have been made for example when an enemy is created it is appended to this sprite group, this is to allow uniform interaction of the enemy instances for example allowing all instances to be referenced simultaneously for reasons such as running their update loop constantly, and allows enemies that have been killed to be removed and no longer referenced so they no longer use client resources. This is the same for the projectiles as well as NPCs.

A 2-Dimensional Dictionary is used as a place holder for the containers accessible by the client and store firstly the keys (tabs of the inventory) and those tabs have their own dictionary containing the item and amount held within, this value is update with the correct items when the client accesses a container.

Next is the use of a Matrice which is treated as a Tree by the A star pathfinding algorithm when the client changes maps the code will use the provided id in the change maps function read the map dictionary return the corresponding map TMX and CSV files, the client will then read and index that csv file into a grid like structure and store that in the Map Classes attribute as path.data. Path data allows more accurate pathfinding as well as adding specific events to certain tiles.

A list is used to store tuples containing the Pygame.rect object of any interacables defined in the tmx file and and is stored alongside its unique ID within that list.

## DATA STORAGE:

Multiple Different files are present within this system:

**Config Directory:**

The config directory contains 3 CSV files the first of which being the enemies.csv file, this file makes use of python's ability to use header identification within a csv file for example, this file contains all the information pertaining to different enemy types such as their stats and images. The header used is as follow: (name,health,speed,damage,cooldown,image,range) very self-explanatory every row beneath the header will follow that syntax and the corresponding data is separated via commas. This file is used client side and server side to construct enemy instances from this data.

Next is the enemyspawns.csv file this file is strictly server side and contains the spawn locations for the enemy ai. Using the header (mapID,name,rectx,recty) as follows ,every row beneath will use the previous enemies file to gather the enemy data and concatenate it with mapID and co-ordinates to spawn that enemy in the game world upon server initialization.

Next is the spawn.csv file which has the exact same header as enemyspawns.csv but contains an additional header for image.png this file is used client side to spawn NPCs at the specified location. The NPCS name and image are used firstly to draw that NPC then for any interaction after the name acts as an identifier for quest completion etc.

**ASSETS DIRECTORY:**

The assets directory is as mentioned is a large storage of PNG files and images used to be displayed while in game. The PNGS are loaded uses pygame.image library and is able to transform, rotate and scale the images as needed. The background music is also found in this file as an mp3 file.

**IMPORTS DIRECTORY:**

The imports directory consists of two .py files.

First of which is the Constants.py file. As the name mentions this is a large python declaration of constants which would otherwise have to be redeclared multiple times in the main client file. Firstly, ctypes is imported in this file and defines the screen\_width and height using the system metrics gathered by the ctypes library, next is the definition of menu sizes using simple / , \* + , - operands to be used by the client when displaying menus or other pygame surfaces. Next is a collection of tuples containing a large variety of RGB colour codes named aptly. This saves lines and memory client side not having to redefine colour codes whenever I'd like to use them. Within the Constants file the path to access other directory's such as assets is kept being concatenated with image paths client and server-side for accessibility.

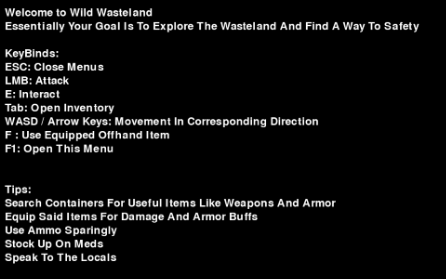
Next is the Items.py File. This file contains 5 dictionaries containign every possible item within the game system. Alongside its data the comments visible at the beginnign of the script indicate how each dictionary is indexed but as a universal basis its indexed as dictionary\_name = {“Item\_Name”: (“ITEM\_DATA[0],ITEM\_DATA[1]...”))}. The reason the items are defined separately is to allow a copy for the client and the server so the server can populate containers using a list of all possible combinations, and client side to display the information of gathered items when viewing the Item.

**MAPS DIRECTORY:**

This directory contains a large amount of data in relation to TMX,TSX and the other tilesets used to display the map. Firstly Software called Tiled can be used to access the contents of a TMX file and is used to build the map by slicing the provided tilesets (whether they be a JPG,PNG or other image format) into tiles each individual tile exists on a separate layers and information about this layer is stored in the name of the layer.TSX and said tiles are assinged an ID which is then converted into a CSV file which using the PyTMX library is read and and pygame is then able to draw the CSV to the screen using the csv and TSX to map each tile and display it to the screen. TMX files also contain objects similar to python the objects that exist within layers can be assigned a dictionary style value for example objects exist on each layer that act as doors,they have a custom property called Interactable and a Value to co-inside with it. Python is able to read these properties and store them as pygame.rect objects. This is how I will manage collisions by utilizing the rectangular properties of the TMX objects and mapping them to Pygame Rects to check for collisions.

## HCI (Controls/Interfacability):

The controls of my project have been made deliberately simple utilizing the universal video games WASD for of movement, however I have also implemented the use of arrow keys to synonymously also be used this could be for left handed users or simply a way to ensure there are two sets of input to increase the robustness of movement in scenarios where the users keyboard may be broken or defective and that play is uninterrupted.



Controls and actions are displayed to the user upon opening the game each key bind is linked to an action explicitly stated to the user, therefore eliminating any concerns of lack of direction a user may experience, alongside the controls are tips on how to interact with the environment and what to do within the game world.



Visbile on the right hand side of the screen is the currently equipped items to inform the player as to what items they have equipped as mentioned in the user inteview that the origianl interface was unclear in informign the player on what items where equipped. Next is the Health bar in the top right originally it was just a green bar set at a ratio to indicate health but as this was too vague a bright blue text was added on the health bar to explicitly indicate the exact amount of health the player had.

Shortly below this is a blue shield with the damage reduction currently gained from the equipped armor, the reason this was added was as before the only way to be informed of the damage reduction was to open your inventory and manually calculate the sum of all damage reductions from Armor you equip which is inefficient.

This also abides by Fallouts style more closely as within that game all player and armor stats present are easily and readily visible via the wrist mounted HUD, or even at times on the screen directly as i have done so far in this design.

## Function List(Server Side):

Beginning with the server –sided functions:

///

read\_csv\_file()

**Parameters:**

csv\_file: This is a string representing the name of the CSV file that the function will read.

**Returns:**

grid: This is a two-dimensional list containing integer values parsed from the CSV file.

**Purpose:**

This function reads a CSV file and converts its contents into a 2D list of integers. It opens the specified CSV file located at the path defined by constants.mappath + csv\_file, reads its contents using the csv.reader, and converts each cell to an integer. The resulting 2D list (grid) is then stored in the maps dictionary with the csv\_file as the key. Finally, the function returns the grid.

///

Sendenemys()

**Parameters:**

id: An identifier for the enemy.

name: The name of the enemy.

rectx: The x-coordinate of the enemy's position.

recty: The y-coordinate of the enemy's position.

hp: The health points of the enemy.

map: The map where the enemy is located.

**Returns:**

This function doesn't have an explicit return. It sends data to connected clients.

**Purpose:**

This function sends information about an enemy to all clients in the CLIENTLIST whose current map (playermap) matches the specified map. It constructs a JSON packet (enemypacket) with the provided enemy details and sends it to the clients. The information includes the enemy's ID, name, position (rectx and recty), health points (hp), and the map they belong to. The function is enclosed in a try-except block to handle any exceptions that might occur during the process.

///

send\_to\_client()

Parameters:

ip: The IP address of the client.

client: The client socket to send data to.

Returns:

This function doesn't have an explicit return. It sends an initialization packet to the specified client.

Purpose:

This function interacts with a database to handle player quest data. It first checks if the player's IP is present in the Player\_Quest\_Data table with QID (Quest ID) as 0. If not present, it inserts a new row with COMPLETION set to 0.

The function then retrieves player information from the Players table based on the IP address. If the player is not found, it initializes the initpacket with default values (X, Y, HP, and map). If the player is found, it populates the initpacket with the player's actual data.

Finally, the function sends the JSON-serialized initpacket to the client socket. The information includes the player's initial position (X and Y), health points (HP), and the initial map (map). The data is sent with a "#" to act as a header and footer for each packet.

///

Draw\_other\_entities()

Parameters:

This function doesn't take any parameters.

Returns:

This function doesn't have a return.

Purpose:

This function iterates over connected clients (connected\_clients) and retrieves player information from the Players table in the database for each client. If the player is found in the database, it constructs a packet (player\_packet) with the player's position (x and y) and assigns it a unique identifier (ID).

It then iterates over all clients in CLIENTLIST and sends the player\_packet to clients that meet the criteria:

The target client's IP is different from the current client (targetip != client).

The target client is on the same map as the current client (playermap[targetip] == playermap[client]).

The packet is sent as a JSON-serialized string , encoded, and transmitted to the target client. Any exceptions during this process are caught and ignored.

The purpose of this function is to allow other player connected to draw other players giving the mmo feel hence why the target ip is never the same as the client ip to prevent the client from drawing themself twice

///

null\_check()

Parameters:

This function doesn't take any parameters.

Returns:

This function doesn't have a return.

Purpose:

This function performs a null check on the Players and Quests tables in the database (database.db). It fetches all column names for each table using the PRAGMA table\_info SQLite statement.

For each column in the Players table, it executes an update query to replace any null values with the string representation of an empty dictionary ('{}').

Similarly, for each column in the Quests table, it executes an update query to replace any null values with the integer 0.

The changes are then committed, and the database connection is closed. The purpose of this function is to ensure that there are no null values in specific columns of the Players and Quests tables which may cause issues later in the code or when a new player with no data joins.

///

quest\_update()

Parameters:

ip: The IP address of the player.

QID: The Quest ID to update.

stage: The new completion stage for the specified quest.

Returns:

This function doesn't have a return.

Purpose:

This function updates the completion stage of a quest for a specific player identified by their IP address. It connects to the database (database.db) and uses a SQL UPDATE statement with placeholders to set the new values for the IP, QUEST, and COMPLETION columns in the Player\_Quest\_Data table.

The function then commits the changes and closes the database connection. If an exception occurs during the process, it is caught and printed to the console. The purpose of this function is to update the quest completion stage for a specific player in the database when they have completed a quest in game.

///

Updateinventory()

Parameters:

client: The client socket for which the inventory is being updated.

Returns:

This function doesn't have a return.

Purpose:

This function updates the inventory information for a specific player identified by their client socket. It connects to the database (database.db) and retrieves relevant data from the Players, Player\_Quest\_Data, and Containers tables.

It populates different packets (Pweaponspacket, Parmorpacket, Pammopacket, Pconsumablespacket, Pmiscpacket, QPACKET, and ContainerLocked) with the corresponding inventory information retrieved from the database.

The function then sends these packets as JSON-serialized strings. The purpose of this function is to update the player's inventory details and send them to the client for display or further processing, for example when a player completes a quest or recieves an item this function is called to ensure the client has the most up to date version of the game data and allows the client to draw that to the user. Any exceptions during this process are caught via the try loop and often discarded. The database connection is closed after the updates are sent to reduce memory overhead.

///

remove\_item()

Parameters:

ip: The IP address of the player.

item\_name: The name of the item to be removed.

amount\_to\_remove: The quantity of the item to be removed.

Returns:

This function doesn't hav a return.

Purpose:

This function removes a specified quantity of a particular item from a player's inventory. It connects to the database and retrieves the current inventory information for the player identified by their IP address.

The function uses JSON deserialization to convert the inventory fields (Weapons, Armor, Ammo, Consumables, and Misc) from the database into Python dictionaries. It then iterates through these dictionaries to find and update the quantity of the specified item (item\_name). If the item is found and the requested amount can be removed, the inventory is updated, and the changes are committed to the database.

If the player is not found in the database or there is insufficient quantity of the item, appropriate messages are printed to the console.

The database connection is closed in a finally block to ensure proper cleanup even if an exception occurs. Any SQLite errors during the process are caught and printed to the console. The purpose of this function is to remove a specified quantity of a particular item from a player's inventory and update the database accordingly.

///

add\_item()

Parameters:

ip: The IP address of the player.

item\_name: The name of the item to be added.

amount\_to\_add: The quantity of the item to be added.

field\_name: The name of the field in the player's inventory where the item should be added.

Returns:

This function doesn't have a return

Purpose:

This function adds a specified quantity of a particular item to a player's inventory in a specific field. It connects to the database and retrieves the current inventory information for the player.

The function uses JSON deserialization to convert the specified inventory field from the database into a Python dictionary. It then updates the dictionary with the specified item and quantity. The modified dictionary is then serialized back to JSON, and the database is updated with the new field value. The reason this add function includes the field name is if the item is not found within the inventory rather than validating every item in the game to see which item is in the correct field it will simply just add the item to the field specified, this is riskier as it requires the field name to be correct so not to accidentally add a weapon to the armor field for example but overall is alot more efficient rather than iterating through every item until I find a match.

If the player is not found in the database, an appropriate message is printed to the console.

The database connection is closed in a finally block to ensure proper cleanup even if an exception occurs. Any SQLite errors during the process are caught and printed to the console. The purpose of this function is to add a specified quantity of a particular item to a player's inventory and update the database .

///

update\_db()

Parameters:

client: The client socket for which the database is being updated.

Returns:

This function doesn't have an explicit return.

Purpose:

This function serves as the main packet handler for updating the database based on various commands received from the client. It continuously receives and processes incoming data from the client socket.

The function decodes the received data and splits it into a list of messages. It then iterates through each message, attempting to deserialize it from JSON. Based on the "COMMAND" field in the message, it performs different database update operations.

Some of the handled commands include updating player positions, initializing and updating inventories, removing and adding items to inventories, handling enemy hits, opening and transferring items between containers and inventories, updating quests, and unlocking containers.

Database updates are performed using SQL queries, and changes are committed to the database. Additionally, the function sends relevant updates or responses back to the client.

The function includes exception handling for handling connection resets and SQLite errors. If a connection reset occurs, the function sends a ping to the client and returns to ensure proper cleanup.

The function runs continuously in a loop, handling incoming data from the client.

///

ping\_client()

Parameters:

client: The client socket to which a ping packet is attempted to be sent.

Returns:

This function doesnt have a return.

Purpose:

This function attempts to send a ping packet to a client. Tt ensures that the client is still responsive and connected to the server . If unsuccessful, it removes the client from various connection-related lists and notifies other clients to kill the player sprite associated with the disconnected client.

The function tries to send a ping packet to the client using the client socket. If the send operation fails (indicating a connection issue), it prints a message indicating the client's disconnection. It then removes the client from the CLIENTLIST, player\_positions, and connected\_clients lists.

It sets the killplayerpacket["ID"] to the ID of the player associated with the disconnected client and sends a kill player packet to all remaining connected clients to remove the player's sprite.

The function handles potential errors during the process, such as removing a client that is not in the list (ValueError) or accessing the client's IP in case of an already removed client (KeyError).

The killplayerpacket["ID"] is reset to "-2" after sending the kill player packet to avoid confusion. The function runs quietly if any exceptions occur during the process. The reason –2 is used as it is never referenced by an actual connected player to ensure players that are still connected are removed from other players screens. -1 is used as an ID for a placeholder as a workaround to python's lack of hoisting.

///

manage\_connections()

Parameters:

This function doesn't take a parameter.

Returns:

This function doesn't have a return.

Purpose:

This function is designed to be run as a separate thread and continuously manages the connections to the server. It iterates through the CLIENTLIST and calls the ping\_client function for each connected client to check their availability.

The function uses a sleep duration of 5 seconds between iterations to avoid excessive resource usage.

The primary purpose of this function is to regularly check the status of connected clients and remove any that may have become unresponsive. It utilizes the ping\_client function to handle the details of the ping operation and client removal in case of disconnection.

///

Enemyhandler()

Parameters:

This function doesn't take any parameters.

Returns:

This function does not have a return.

Purpose:

This function represents an enemy handler that continuously manages the behavior and updates of enemy entities in the game.

It iterates through the list of enemies and checks if each enemy is alive. For alive enemies, it updates their position and status based on the positions of players and the maps players are on.

The function includes a mechanism (ForceEnemyUpdate) to ensure that enemy updates are sent at a regular interval and that enemys that have not yet began pathfinding are visible to the player. If the elapsed time since the last update exceeds 0.5 seconds, it sends an update packet for each enemy to all connected clients using the sendenemys function.

The check\_nearest method is called for each alive enemy, which involves determining the nearest player and updating the enemy's behavior accordingly. The purpose of this function is to handle the continuous updates and behavior of enemy entities in the game. The update interval ensures that enemy positions are regularly broadcasted to clients.

///

spawn\_enemies()

Parameters:

This function doesnt have a paramater passed to it as the CSV file path is hard-coded within the function

Returns:

This function doesn't have a return

Purpose:

This function reads enemy spawn data from a CSV file (enemyspawns.csv) and uses it to spawn enemy entities in the game. Each row in the CSV file represents an enemy spawn with information such as the map ID, enemy name, and initial position.

The function opens the CSV file and uses csv.DictReader to read its contents. It then iterates through each row, extracting information such as mapID, name, rectx, and recty. It creates an instance of the Enemy class using the Enemy.from\_csv method, passing the extracted information.

The purpose of this function is to read enemy spawn data from a CSV file and spawn enemy entities in the game. The printed message indicates the number of active threads, for debugging or monitoring purposes.

///

Saturatecontainers()

Parameters:

This function doesn't take any parameters.

Returns:

This function doesn't have a return.

Purpose:

This function populates the item containers in the game with random items and quantities based on predefined ranges. It reads container data from the Containers table in the database, selects random items from predefined categories (Weapons, Armor, Ammo, Consumables, Misc), and assigns random quantities to them.

The function first accesses the Containers table, fetching data related to container ID, Weapons, Armor, Ammo, Consumables, and Misc.

It uses a dictionary to map category names to the corresponding item dictionaries. The quantity\_ranges dictionary specifies the quantity ranges for different item categories.

The function then iterates through the container data, and for each container and category, it selects a random item from the item dictionary, assigns a random quantity within the specified range, and updates the container's item dictionary with the new data.

Blacklisted items, such as None in the "Ammo" category and "Trophy" in the "Misc" category, are handled to prevent them from being added, this may be due to being quest items or just that i dont want them to spawn naturally.

///

Lockcontainers()

Parameters:

This function doesn't take parameters.

Returns:

This function doesn't give a return.

Purpose:

This function determines whether each container in the game world should be locked or unlocked based on a calculated lock chance. The lock chance is influenced by the container's ID, with higher IDs having a higher chance of being locked. The lock chance is capped at 75%.

The function accesses the Containers table in the database to fetch the IDs of all containers.

It then iterates through each container, calculates the lock chance based on its ID, generates a random number between 1 and 100, and compares it with the lock chance. If the random number is less than or equal to the lock chance, the container is set as locked (Locked = 1); otherwise, it is set as unlocked (Locked = 0).

The purpose of this function is to introduce variability in container lock status, making some containers more likely to be locked than others based on their ID mainly for progression so items become harder to find the more progressed into the game you are, sharply raising the difficulty curve and making sure that players have to use items sparingly.

////

## Function List(Client Side):

Now For the Client Sided Functions:

recv\_server()

Parameters:

This function doesn't take any explicit parameters.

Returns:

This function doesn't have an explicit return.

Purpose:

The purpose of this function is to actively and independently receive data from the server. This function is executed in a separate thread, and it continuously listens for incoming data from the server.

The received data is processed and interpreted based on the "COMMAND" field in the JSON packet. Depending on the command, different actions are taken:

The data is decoded, split into individual packets using the "#" character as a delimiter, and then processed one packet at a time.

Exception handling is present to catch any errors during the data processing.

This function is designed to run continuously in a separate thread to handle server communication independently from the main game loop.

///

Drawotherplayers()

Parameters:

sid: ID (sprite id) of the other player.

RECTX: X-coordinate of the other player's position.

RECTY: Y-coordinate of the other player's position.

Returns:

This function doesn't have an explicit return.

Purpose:

The purpose of this function is to update the position of other players on the client's screen based on the received data. It is responsible for drawing and updating the positions of other player sprites.

Functionality:

The function takes the sid (player ID), RECTX (X-coordinate), and RECTY (Y-coordinate) as parameters.

It iterates through the existing otherplayers sprites to find the sprite with the matching sid.

If the sprite with the given sid is found, it updates the sprite's position (rect.x and rect.y) based on the received coordinates (RECTX and RECTY).

If the sprite with the given sid is not found (indicating a new player), it creates a new OTHERPLAYER sprite with the provided sid, RECTX, and RECTY and adds it to the otherplayers group.

This function ensures that the positions of other players are accurately updated on the client's screen.

///

get\_object\_rects()

Parameters:

filename: The filename of a Tiled map file (TMX file).

Returns:

A list of pygame.Rect objects representing the rectangles of objects present in the Tiled map.

Purpose:

This function aims to extract the rectangles of objects from a Tiled map file.

The function utilizes the pytmx library to load the Tiled map data from the specified filename.

It initializes an empty list named object\_rects to store the rectangles of objects.

It iterates through the visible layers of the Tiled map.

For each object group layer (pytmx.TiledObjectGroup), it iterates through the objects in that layer.

For each object, it creates a pygame.Rect object using the object's position (obj.x, obj.y) and dimensions (obj.width, obj.height).

The created pygame.Rect objects are added to the object\_rects list.

Finally, the list of pygame.Rect objects representing the rectangles of objects in the Tiled map is returned.

///

get\_interactable\_rects()

Parameters:

filename: The filename of a Tiled map file (TMX file).

Returns:

A list of tuples, where each tuple contains:

Interaction type (interType),

pygame.Rect object representing the rectangle of the interactable object (interRect).

Interaction ID (interID).

Purpose:

This function aims to extract information about interactable objects from a Tiled map file.

It initializes an empty list named interactable\_rects to store information about interactable objects.

For each object, it checks if the object has a property named 'interactable' set to True. If yes, it proceeds to process the object.

It extracts the properties 'intertype' and 'interactable' from the object, creating variables interType and interID.

It creates a pygame.Rect object (interRect) using the object's position (obj.x, obj.y) and dimensions (obj.width, obj.height).

It appends a tuple containing interType, interRect, and interID to the interactable\_rects list.

Finally, the list of tuples representing information about interactable objects in the Tiled map is returned.

///

draw\_health\_bar()

Parameters:

screen: The Pygame display surface where the health bar will be drawn.

player: An object representing the player, containing information about health (hp), maximum health (max\_hp)

Purpose:

This function is designed to draw a health bar on the Pygame display surface to represent the player's health.

The function first defines the sizes and positions of the health bar components, such as width, height, padding, and health ratio. It calculates the width of the health bar based on the health ratio and draws the current health portion using a green rectangle. It creates text surfaces for the damage reduction value and player's current health.

///

draw\_equipped\_items()

Parameters:

screen: Pygame display surface.

player: Player insatnce

Returns:

None. (The function directly modifies the display surface.)

Purpose:

The purpose of this function is to visually represent the equipped items of the player on the Pygame display. It iterates through the equipped slots, drawing a black square for each slot. If an item is equipped in that slot, it retrieves and draws the corresponding item image on the display. This function contributes to providing the player with a visual representation of their currently equipped items in the game.

///

spawn\_NPCS()

Parameters:

map: A string representing the map ID where NPCs are to be spawned.

Returns:

None.

Purpose:

The purpose of this function is to read NPC spawn data from a CSV file and spawn NPCs on the specified map. It iterates through the rows of the CSV file, checking if the mapID matches the provided map parameter. If a match is found, it extracts information such as the mapID, NPC name, position, and image path, and uses this information to create an NPC object. The function is responsible for spawning NPCs in the game world based on the provided map.

///

Cleanup()

Parameters:

None.

Returns:

None.

Purpose:

The purpose of this function is to clean up the game state by removing defeated enemies. It iterates through enemysprites, which represents a collection of enemy sprites, and checks if the health of each enemy is less than or equal to 0. If an enemy is defeated, it removes the enemy from the enemies collection, sets its alive attribute to False, and removes the enemy sprite from enemysprites. This ensures no dead enemies are drawn to the game surface

///

OpenCont()

Parameters:

ID (int): The ID of the container to be opened.

Returns:

None

Purpose:

The purpose of this function is to handle the opening of containers in the game. It checks if the specified container ID is in the lockedcontainers list, indicating that the container is locked. If the player has a "Lockpick" in their inventory, it deducts one from the amount of lockpicks, initiates a lockpicking mini-game using the LockpickingGame class, and sends a corresponding update to the server using the RMV packet. If the player doesn't have a lockpick, it displays "LOCKED" on the screen using screentext.changetext. If the container is not locked, it sends a CONT packet to the server and initiates a transfer game using the Transfer class. The function effectively manages the opening of containers in the game and handles locked containers being interacted with.

///

Changemaps()

Parameters:

id: The ID of the map to which the player is transitioning.

Returns:

Doesn't have a return per-say but changes the map level for the client.

Purpose:

The purpose of this function is to handle the transition of the player to a new map in the game. It updates the player's current map ID, loads the corresponding TMX and CSV files for the new map, and creates a new Map object. It then calls the changemap method of the Map class to perform the necessary actions for transitioning to the new map, such as updating the display, handling NPCs, and managing interactables.

///

update\_camera()

Parameters:

player: The player instance representing the in-game character.

Returns:

A tuple containing the X and Y coordinates representing the camera offset.

Purpose:

The purpose of this function is to calculate and update the camera offset based on the position of the player within the game world. The camera is adjusted to keep the player centred on the screen. The function takes into account the screen dimensions, player position, and the size of the game map. The calculated X and Y coordinates are then constrained within the limits of the game map to prevent the camera from going beyond its boundaries. The updated camera offset is returned as a tuple. This offset isused later in projectiles and the render function to accuratley draw the game state to the players screen.

///

Render()

Parameters:

screen: The Pygame screen surface where the game is displayed.

map\_surface: The Pygame surface representing the game map.

camera\_x: The X-coordinate offset of the camera.

camera\_y: The Y-coordinate offset of the camera.

player: The player object representing the in-game character.

Returns:

This function does not return any value it simply updates the screen

Purpose:

The purpose of this function is to render the game elements on the screen. It adjusts the player's position relative to the camera, draws the map surface and the player on the screen, and then resets the player's position after rendering. Additionally, it calls functions to draw equipped items on the screen and handles displaying text messages. This function contributes to updating the visual display of the game during each frame.

///

load\_equipped()

Parameters:

player: The player instance representing the in-game character.

Returns:

This function does not return any value.

Purpose:

The purpose of this function is to load equipped items for the player from a JSON file. It reads the JSON file containing information about the player's equipped items, updates the player's equipped slots accordingly, and loads corresponding images for equipped items. The function handles different types of equipped slots, such as head, chest, feet, hand, and offhand. This function is called upon the player loading into the game once the player loads it checks the items that were previously equipped still exist in the inventory if so automatically equips them and updates the damage reduction statistic to prevent the player from having to reequip every item each time they load in.

///

Save()

Parameters:

None.

Returns:

None.

Purpose:

The purpose of this function is to continuously send player data to the server for saving. It operates in an infinite loop and periodically sends the player's position (X and Y), health (HP), rectangle coordinates (RECTX and RECTY), and current map (Map) to the server. This information is sent as a JSON-encoded string with the command type "GENERAL." The function introduces a delay of 0.5 seconds between each transmission to avoid overwhelming the server with frequent updates. This function is crucial for maintaining the player's progress and state on the server side. This function is called via a thread later once the player has successfully loaded into the game to ensure an accurate representation of the player is always recorded server side.

///

Playerinit()

Parameters:

None.

Returns:

None.

Purpose:

The purpose of this function is to initialize the player. It starts by checking if there are existing player coordinates (xpos, ypos) and health (hp), these would be recieved from the server via the rec\_server thread called before this function. If these values exist, it creates a Player instance with the given coordinates and health, and sets the global variables player, pmap, and hp accordingly. If the values are not available, it initializes a default Player instance at coordinates (0, 0) and on map 0. It then sends an initialization command to the server using the INIT packet. Subsequently, it starts a new thread to continuously save the player data using the save function. Finally, it transitions to the main game loop after changing maps and running the main menu.

///

Main()

Parameters:

player: An instance of the Player class.

Returns:

This function does not return any value.

Purpose:

The purpose of this function is to serve as the main game loop. It manages the continuous execution of the game logic, rendering, and event handling. The loop includes the following key functionalities:

Updating player and other game elements (NPCs, enemies, projectiles).

Handling collisions between enemies and the player.

Drawing the map, health bars, and other visual elements.

Monitoring events, such as key presses and window closure.

Sending player data to the server.

Exiting the game safely when the window is closed.

The main loop is called via the player init function to ensure the game only begins when all the relevant information has been received from the server, the main loop also initializes the pygame clock tick speed limiting the frames per second to 60 for performance and stability.

### Startup.py:

create\_tables()

Parameters:

player: An instance of the Player class.

Returns:

This function does not return any value, but creates a file and database tables if needed

Purpose:

The purpose is to first identify is the database.db file exists if so checks if all of the table files are integral. If they are not then the program will build them from the hardcoded schema

Once run it will also begin a server after verifying the database has been created successfully

Also at the beginning of the code PIP is used to install some prerequisites

## File Structure:

My project is comprised of many different files all providing a unique and important properties to the gameplay, so it is important to understand how they are structured. The game will be transported via a zip file.

Existing in the root of the directory are dependant whether you're an average user or the developer for example in the main root for players will only exist the game.py (client file) as well as the equipped\_items.json file, these are in the root as they are the most utilized files and are not a prerequisite to any other files.

Within the root directory there are 5 directories' being additionals,assets,config,imports,maps although the names are self-explanatory, the additional file is a developer only and hosts additional files such as python scripts to quickly drop records or reset the database and won't be accessible to the user.

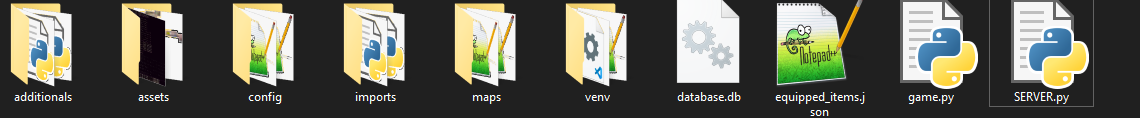
Next is the Asset directory which holds all the PNG assets used withing the game alongside all the sprites and other media that may appear in the game when running it.

The Config directory will exist both for the developer and user however its contents will not be identical as client side it holds the NPCs spawn locations , names and image paths as well as the enemy information like name, attributes and file path, whereas the developer side also has access to the enemyspawns.csv which can be altered to spawn new enemies or change the positions at which ones spawn.

Imports exists for both users and devs and contains to .py files these being constants which hold arbirituary values like colour tuples or screen dimensions as well as values that just do not change such as the server ip and port. Also, present is the items.py file which hosts every item in the game alongside their stats as well as their image paths.

Finally, is the map directory which also hosts all the files pertaining to maps such as the tmx,tsx and other files needed to construct the maps which are utilized in game.

The database file and server.py file is also present in the root directory but only on the developer side as users will be connected to one server and that database will be modified accordingly.



Additionally since writing the above documentation a new file called startup.py also exists in the root directory, which is used to build the database file and host a server for first time users.

## Key functions:

Server Side:

Main- The main function firstly begins all of the threads serveside these being, manage\_connections,draw\_other\_entities, enemy handler next he main function of the server contains a stopping function that sets up a socket connection on a specified port found in the constants file , it then binds this connection and listens for incoming connections from clients. This means sockets are vital to allow simultaneous occurrences while the server is stopped waiting for new incoming connections, once connected the server runs a null check and appends the clients socket to client list and ip to connected clients list for later reference, the function also uses the enumerate function to assign each client an ID which will be used to display themselves to other clients, once the connection is solid and has been validated it then begins the update\_db thread which handles all network traffic from the client.

Read\_csv\_file: This function as predictable reads a privded csv file and converts these into a grid matrice and then stores this new grid in a dictionary with the name of the map as the key, this is done at the start of the code and all maps are loaded, this is because enemy instances are handled server side and hence so is there pathfinding, the grid being a vital part of their tracking algorithm must be created before the enemies are to prevent conflicts with the code and ensure no matter what map a player or enemy is on there is suffice information to create a smooth experience.

Send\_enemys: This function simply iterates throught the clientlist mentioned earlier and sends the data of an enemy to that client, it takes the parameters like ID, Health e.t.c and updates a dictionary called enemy packet with the data and then uses json the serialize the data into a byte array for transmission via sockets.

ENEMY CLASS(

Firstly the enemy class itself is complex as instances are not created via the use of init function as most classes were instead a class method is used in combination with set constants to create the instance.

From\_csv: A csv is used to store the enemies data, this contains their name their health, speed, damage, the function is passed the enemy name, the map and the x,y coordinates, the function then will read the enemies csv gather that additional information related to that enemy and then finally return an instance of that enemy, within the init block however the enemy is added to a sprite group called enemies for it to be referenced later on.

Check\_Nearest:

This function firstly checks the enemy referenced is alive is so it sets the closest player and distance attributes to none. What it will then do I iterate through the dictionary containing all players on the enemies map and their position, a tuple containing the players x and Y are created and using pythagerous the enemy works a distance from itself to the player, the client with the shortest distance is determined to be the target, the enym will then check if that distance is below a certain threshold set for each enemy and if so will then utilize the provided martice from the map, its selfs position and the players position and utilize the A star pathfinding algorithm to create a list of tuples referring to the position in the matrice it must take to reach the client, while this calculation is made the pathing file containing areas the enemy can not walk is considered to allow smooth movement client side. Once a path has been found the move function is called

Move(): Move uses a time clause to ensure the enemy does not instantly teleport to its new position and first checks its position against the position of the path and then moves accordingly for them to match. The time clause is updated on each full movement cycle, the distance in which the enemy moves is also determined from the csv in its creation.

)

Send\_to\_client():

The send to client is the intilaizing block for clients which uses SQL to gather all the relevant client data serialize it and send it.

Null\_check():

Null check is a special function I created with defensive design in mind to protect the elements of the database from having null values caused by invalid movements of items. The SQL Pragma function is used to gather the metadata from the tables and iterate through that specific table info, the code will check for any null occurrences and overwrite them with a blank dictionary, likewise for quest data this is to ensure quest data or inventory data can never be null as multiple functions use comparison against these pieces of data and a null value would cause exceptions, it also acts as a way to ensure new players who have no existing data in the database have a blank slate and have no null values. I understand the use of NOT NULL could be used when creating the records but when updating and deleting json dictionaries from the database the added check when interacting with items or quests provides extra protection against unexpected exceptions.

UpdateDb:

# Testing:

## TEST STRATEGY:

The project will be tested to ensure the functionality and accessibility of all components of the game. Any that don't adhere to the expected outcome will be rectified for the final version to be presented. Below will be constructed a table containing a test plan to index the tests and group them into similar categories, alongside the table will be a narrated video in which the table contains timestamps pertaining to the section which demonstrates that bespoke function working as intended.

To begin testing i will clear all records from my database, to ensure the experience is that of a new user utilizing my program. Once connected to the game i will show the databased current state to demonstrate the database is accurately recording and updating the players data. Once i have connected i will move my character and once again show the database which will reflect the coordinated being updated to reference player movement.

Once connected I will open the inventory to present a blank inventory and navigate the game world until i arrive at a place to gain items, this will be a container. Once the container is opened and items are being transferred i will once again show the database to reflect how the different tasbles are being updated to accomplish the seamless movement of items.

Once a weapon has been equipped i will demonstrate the weapon will not fire if the correct ammunition is not also present within the user inventory, once this has been completed i will collect the required ammunition and show the differing fire rates of the weapons within the game.

After this I will equip a piece of pieces of armours to show how the equipped slot status reflects the items currently in use by the player as well as show how the GUI has updated to reflect the newfound increase in stats.

Next up will be enemy pathfinding, within the game world multiple types of enemies will be present each harbouring their own stats, I will demonstrate how an enemy will path find towards the player and around unwalkable objects like walls or lakes, and how defined regions such as the spawn area are unavailable for enemies to traverse due to this.

Then to demonstrate the weapons further I will show how bullets and projectiles fired from the weapon inflict damage to the enemies and the visual indicator associated with this. Furthermore, is the process in which when an enemy is killed a reward is provided to the player in terms of currency.

Next is the traversal of different map layers and the independence of containers, NPCS and enemy's alike showing how the player can traverse different map layers, on the map layer will be an NPC which will be interacted with and display a quest to the user in which I will attempt to complete.

The map layer i will be showing contains 2 containers each with their own inventories i will access the contents and once again show how the database reflects the contents accurately, I will then show how taking an item and storing an item also reflects the change in the database.

I will then turn in the quest item to the NPC and show how rewards are provided to the player upon completion of a quest. As well as how the quest objective is updated in the database.

I will the traverse the map to demonstrate the cameras scrolling ability and render function always keeping the player on screen and offsetting the map to accommodate for the player moving.

I will then encounter another NPC surrounded by additional enemies and will also complete the quest proposed by this NPC as to prove the proof of concept.

I will then demonstrate the lockpicking minigame for containers which is a heavy mathematical sub program which allows containers to be unlocked. Once again, the database will be shown to reflect the lock status changing.

Multiplayer will be demonstarted by an additional client joining the game and their movement and sprite will appear on the other client's screen, this new player will only appear on the same map as u to prevent overlay issues and they will have their own inventory. To ascertain to the multiplayer style this other player will also be able to interact with enemies and damage or even kill enemies present on our own screen. As the enemies are server side, they will also path find to the other client which will be shown.

Finally, the death screen and other UI/GUI features will be show as well as the replay ability.

## TEST PLAN:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Purpose | Expected Outcome | Outcome | | Test Data | TimeStamp |
| 1.0 | Demonstrate simultaneous player movement | The ability for the player to move in 4 cardinal directions and change X,Y position simultaneous | As expected | | Shown | 1:40 |
| 1.1 | Demonstrate a circular queue iteration to show an animation loop | As the player moves the animation changes and iterates simulating walking | As expected | | Shown | 20:47 |
| 1.2 | View a second connected client in game | A sprite will appear map independent and will show the other player moving | As expected | | Shown | 21:22 |
| 1.3 | Demonstrate collision between objects in the game world | The player should not be able to pass/and or traverse through an object | As expected | | Shown | 13:38 |
| 2.0 | The use of a camera to render the world as well as the player | The player should appear on top of the map layer | As expected | | Shown | 15:28 |
| 2.1 | Demonstrate players sprite permeance | The player should always be visible on the screen | As expected | | Shown | 15:36 |
| 2.2 | Camera Scrolling | The camera should scroll along the map to keep the player visible and render the new parts of the map. | As expected | | Shown | 15:30 |
| 2.3 | On screen text | Any uI elements should be drawn on the screen to indicate ability for interaction | As expected | | Shown | 3:20 |
| 3.0 | Converting Objects created within a tmx to Pygame instannces | Objects defined within the TMX layer should be converted in collidable objects in the game world | As expected | | Shown | 24:20 |
| 3.1 | Testing presence of object data | When interacting with containers clients should dbe able to read the container ID | As expected | | Shown | 5:04 |
| 3.2 | Distingushing Object types | Objects should be identified as either container doors or objects | As expected | | Shown | 3:20,13:30 |
| 3.3 | Intercation | The player should be able to interact with these objects | As expected | | Shown | 13:30,13,48 |
| 3.3.1 | Containers | Accessing a container should open an inventory window with the items stored in that container to the player. | As expected | | Shown | 3:23 |
| 3.3.2 | Doors | Doors should change the player map dependant on the ID of the door. | As expected | | Shown | 13:30 |
| 4.0 | Client Server Connection | The client and server should be able to interact with one and other using packets via a socket connection. | As expected | | Shown | 20:18 |
| 4.1 | Json Serializing | Data packets should be serialized into byte data for transmission and then decoded when received to produce a dictionary | As expected | | Shown | 25:05 |
| 4.2 | Packet Identification | Packets should be handled dependant on their command payload and are separated by ‘#’ that act as header. | As expected | | Shown | 25:30 |
| 4.3 | Server Load | Test if any packet loss or delay occurs when multiple sockets are connected | As expected | | Shown | 20:30 |
| 5.0 | Enemy Instances | Gather the required parameters bout enemy spawns from a CSV file and pass it to a class method | As expected | | Shown | 18:25 |
| 5.1 | Enemy Instance Creation | Use the provided data in combination with another csv to gather the required parameters for an enemy to spawn | | As expected | Shown | 26:36 |
| 5.2 | Enemy drawing | Enemies should be drawn alongside their health bar on the map surface | | As expected | Shown | 18:15,11:15 |
|  |  |  | |  |  |  |
| 5.3 | Enemy Tracking | The enemy should always locate the closest player that is on their map and set them as the target using Pythagoras from a mapped dictionary | | As expected | Shown | 9:43 |
| 5.4 | Enemy pathfinding | The enemy should used the matrices provided by the map to path find around non walkable tiles to the target. | | As expected | Shown | 10:41 |
| 5.5 | Enemy movement | Using the path generated server side enemies should follow this path to the player by changing their x,y values | | As expected | Shown | 10:41 |
| 5.6 | Interpolation | Interpolation should be used to smooth the enemy movement client side ensuring the enemy does not teleport from spot to spot | | As expected | Shown | 9:50 |
|  |  |  | |  |  |  |
| 6.0 | Database | Store data pertaining to players in an SQL client | | As expected | Shown | 4:12 |
| 6.1 | Player saving | The player or client instance should periodically send the server data about the client to act as saving. | | As expected | Shown | 4:12 |
| 6.2 | Database sanitation | The database should be periodically sanitised of all NULL values to ensure | | As expected | Shown | 27:47 |
| 7.0 | NPCS | Ensure NPCS are visible to all players dependant on their map and quest progression | | As expected | Shown | 14:07 |
| 7.1 | NPC Quests | Allow the NPCS to be spoken to with quest sepicifc interaction dialouge | | As expected | Shown | 14:07 |
| 8.0 | Minigames | Test that containers that are locked cannot be accessed without completing lockpicking minigames. | | As expected | Shown | 23:22 |
| 9.0 | Inventory Management | Test the inventory screen accurately reflects the players inventory with their items e.t.c | | As expected | Shown | 5:56 |
| 9.1 | Inventory Accesibility | Ensure the inventory can be interacted with and items can be equipped | | As expected | Shown | 5:56 |
| 9.2 | Equipped Item Persistance | Testing that items that were previously equipped are auto equppied when re loading into the game | | As expected | Shown | 17:29 |
| 10.0 | Weapon Functionality | Ensure Weapons can be used, fired | | As expected | Shown | 6:00 |
| 10.1 | Weapon types | Ensure the weapon types are coherent with their types e.g Guns fire bullets,Melee is for CQB. | | As expected | Shown | 5:40 |
| 10.2 | Ammo Reduction | Ensure the client consumes the correct amount of and type of ammo for specific guns. | | As expected | Shown | 6:25 |
| 10.3 | Bullet Types | Ensure different calibers have different speed and damage | | As expected | Shown | 7:03 |
| 10.4 | Reloading | Ensure the weapons can be reloaded with their correct ammo | | As expected | Shown | 5:50 |
| 11.0 | Handling Player Death | Ensure the entity that kills the player is shown and the player is sent back to the origin of the game. | | As expected | Shown | 17:55 |
| 12.0 | Threading for autosave features | Autosave threads should periodically send player data to the server | | As expected | Shown | 29:11 |
| 13.0 | Exception handling | Ensure no unreachable code, handle server disconnects and database errors | | As expected | Shown | 30:0 |
| 14.0 | External files usage | Verify the game utilizes external files for variables, constants, and assets | | As expected | Shown | 31:54 |
| 15.0 | Menu for keybinds/instructions | The menu should display keybinds and instructions for easy reference | | As expected | Shown | 1:17 |
| 16.0 | Server-side container saturation | Containers should be filled with items based on a server-side script | | As expected | Shown | 0:30 |
| 17.0 | Server-side rewards for killing enemies | Players should be rewarded with currency for killing enemies | | As expected | Shown | 11:28 |
| 18.0 | Quest management with sets | Ensure only one quest of a certain type can be active at a time | | As expected | Shown | 16:20 |

Evaluation

|  |  |  |
| --- | --- | --- |
| OBJECTIVE | MET Y/N | ADDITIONAL COMMENT |
| Player Movement in 4 Cardinal directions | Y | Can move along X,Y simultaneously |
| Draw a map to the screen using tile sets | Y | Tiled and PYTMX aided in translating the data to pycharm |
| Object Intercation | Y | Objects can be intracted with and some have quest specific requirements |
| Collisions | Y | Collisions work for both players and enemies |
| Socket Connection | Y | The server can handle multiple clients assigning a thread to each new client that joins. And is able to transmit data to and from the server |
| Enemy Instances | Y | The pathfinding algorithm A\* finder is used server side and that pathing data is relayed to relevant clients. |
| SQL Database | Y | Multiple tables within the database are read accessed and updated to store player data as well as environment data. |
| Multiplayer | Y | The client server model will allow multiple clients to join to the server simultaneously and other players will be drawn to the screen |
| NPCS with dialouge | Y | NPCS have been implemented to act as quest givers and can be spoken to these to have specific quest requirements. |
| Containers | Y | There are functioning containers with multiple items that can be accessed simultaneously by clients. |
| Item system | Y | Items can be accessed via containers displayed in the inventory and equipped by the player |
| Mini-Games (Lockpicking) | Y | Code been fully implemented and is fully functional as of the most recent version. |
| Inventory | Y | The inventory is able to be accessed via the tab key and can switch tabs to sort items into an easier viewing window. |
| Item dependant Values | Y | External files containing the data and stats are used to assign the weapons and Armor values that are used in game as either damage fire rate or Armor. |
| Map Switching | Y | Maps are able to be switched via the use of doors which can be found in the game world and have map specific enemies or NPCS located within |
| UI/HUD | Y | Hovering over the player displays their health bar as well as the on screen text appearing when information is provided to the player such as prompts or dialogue. |
| Threading | Y | Threading exists server and client side to allow the player to receive packets of data irrespective to the current in game situation and vice versa. |
| Exception Handling | N | There are many Exception catches using try but a server crash failsafe needs to be added to the client |
| External File use | Y | External file types are used such as csv, tmx,py files |

## TEST VIDEO:

<https://youtu.be/3wfY-foaOyo>



# EVALUATION

### Future Improvements:

**QOL improvements:**

I believe that future improvements that could be implemented is refencing the objects such as containers and doors as classes and using inheritance to assign them their features rather than storing their specific data within lists.

Next If I was able, I would’ve fully mapped each map in terms of pathfinding as the map never changes so caching paths used by enemies to get from one position to another would save resources and times.

Next, I would make the enemy threads less intensive as they consume a lot of processing time performing mathematical calculations like distance calculation then pathfinding algorithms.

**Dungeons:**

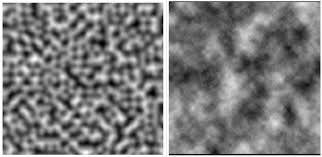
One improevemt to be made pertains to the criticism received from my end user and that is the ability of replay ability and content. In response a sure-fire way to improve replay ability and keep content fresh is the use of procedural generation. Specifically, to create dungeon esc occurrences that spawn randomly throughout the world that can be accessed by players to enter their own specific procedural dungeon containing randomized layout, loot, and enemies. Also containing a boss enemy to fight.

The way I would accomplish this is the use of Perlin noise a strategy to generate a random and procedural pattern, then I would apply a grid over the top and assign each grid a binary value based on the monochromatic RGB values. The values will represent rooms which will be designed and stored client side and then concatenated and arranged to form a large vault style layout. To ensure completability A star pathfinder, which is already in user, will be used to calculate if there is a valid route from the entrance to the exit. If not, a new structure will be regenerated until there is.

Each of these rooms will contain specialised tiles which will either spawn a random enemy or container containing loot, (values can be assigned to containers to dictate the rarity and amount of loot). Overriding will be used to spawn the boss which will be a variation of an existing enemy that's stats will be increased via the use of overriding.



Concept art from Fallout shelter adopting this style.



Time Dilation:

Similar to fallouts style an ability to slow client-side time and allow more accurate shots at enemies and provide breathing time in tense situations, a way to incorporate this already exists within the project but lies unused.

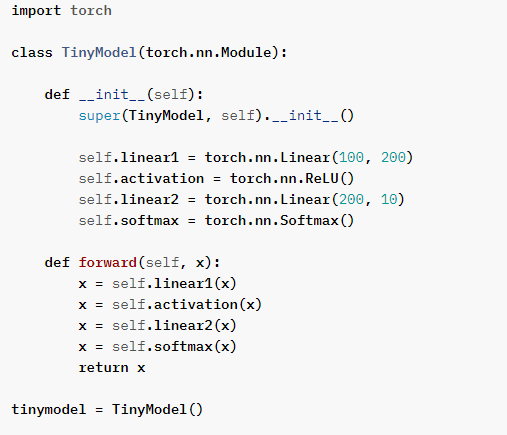
Simply the pygame clock tick is multiplied by a constant knows as the time factor, which is defaulted to 1, limiting the pygame clock will slow client side by making each game loop longer giving the slower time appearance. This will not impact the server at all or other players only the client.

**Enemy AI:**

One major improvement that will be considered for future iterations is the use of Py torch to create an AI for the enemies which includes a variety of states, such as fleeing, healing, using projectiles and weapons.

So far, all the enemies within the game are melee so to add enemies with ability to use deep learning and machine learning this will drastically increase the difficulty curve and and provide a more dynamic and engaging combat experience for the user.

As mentioned, the way this will be completed is via the use of py torchs library and their deep learning functions to build a model alongside the existing game variables that will be able to interact modify and utilize the game world to their advantage, below is a proof of concept utilizing py torches syntax.



The only issue with using Py torch is this is very computationally expensive and having large amounts of enemy's id need to build only one or two models and assign all enemies to use that model to prevent resources being consumed and inefficiencies beginning to form.

# TECHNICAL SOLUTION:

## Complex Functions:

The first complex function is the null check function, essentially what the null check does is utilize the pragma instruction to scrape all the information from the database from all fields and search for any Null values, it will then replace the Null with a default value to prevent any errors during runtime. The reason i consider Null\_Check a complex function is it acts as an aggregate SQL function joining all the tables and metadata into one lump data source and then extracting the required data from it using parameters provided, i believe this alligns witht the group A skills hence its inclusion. This function is found at line 276 in server.py.

Next is the Lockpickgame.Run() method for the Lockpicking class, the reason for its inclusion si the mathematical operations performed to utilize the players mouse position as well as the rotation angle of the lockpick to determine the current angle at which the lockpick is orientated, the reason i have included this is due to it falling under a Group B skill of mathematical calculations updated and performed in real time. It is located in line 1326 of game.py

A major function create\_tables() I believe should be included is the function which validates the database exists and contains all the valid tables, it does this by using the Group A skill of cross-table paramaterised SQL statements to access the sqlite\_master schema table and ensures the table exists using hardcoded named parameters. If the table does not exist then one is created using a set SQL statement, this is repeated for every table and is vital to ensure error handling in case of corruption or loss of the table or database entirley. This Function can be found in startup.py on line 7. After completing this function the program will automatically attempt to host a server.

# CODE:

## game.py:

1: #Imports neccesary librarys

2: import pygame

3: import pygame.sprite

4: import pytmx

5: import csv

6: import sys

7: import random

8: import socket

9: import threading,json

10: import time

11: from pygame import mixer

12: import ctypes

13: import math

14: from imports import constants #imports constants from a fiel to be used

15: from imports import items #imports a list of existing items to gather infor and values

16: user32 = ctypes.windll.user32

17: pygame.init()

18:

19: #Tests if the device has suitable audio output if not ignores and continues

20: try:

21: mixer.init()

22: except:

23: pass

24:

25: hostname = socket.gethostname()

26: client\_address = socket.gethostbyname(hostname)

27: client\_address = '192.168.0.117'

28: COUNTER = 0

29: fpressed = False

30: font = pygame.font.SysFont(None, 20)

31: screen\_width = int(user32.GetSystemMetrics(0))

32: screen\_height = int(user32.GetSystemMetrics(1))

33: screen = pygame.display.set\_mode((screen\_width, screen\_height))

34:

35: #PACKETS FOR THE CLIENT TO SEND TO THE SERVER

36: DATA = {"IP": client\_address,"X": "0", "Y": "0","HP": "0","RECTX" : "0", "RECTY": "0", "Map": "None", "COMMAND": "GENERAL"}

37: PING = {"IP": client\_address, "COMMAND": "PING"}

38: INIT = {"COMMAND":"INITINV"}

39: ENEMYHIT = {"COMMAND":"EH","ID":"-1","DMG":"0"}

40: CONT = {"COMMAND":"OC","ID":""}

41: TOCONT = {"COMMAND":"TC","TAB":"Weapons","ID":"0","Item":"Gun","A":"1"}

42: FROMCONT = {"COMMAND":"FC","TAB":"Weapons","ID":"0","Item":"Gun","A":"1"}

43: QUESTUPDATE = {"COMMAND":"QU","ID":"","VALUE":""}

44: RMV = {"COMMAND":"RMV","AMMO":0,"AMOUNT":""}

45: ULK = {"COMMAND":"ULK","ID":0}

46:

47: s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) #CREATE A SOCKET WITH THE SERVER

48: s.connect((constants.IP\_ADDRESS,constants.PORT)) #CONNECT TO SAID SOCKET USING A PORT DEFINED IN CONSTANTS

49:

50: map\_dict = {"0":("pygame.tmx","test\_PATHING.csv"),

51: "1":("map2.tmx","map2\_PATHING.csv")} #HERE A DICTIONARY IS USED TO STORE MAPS AS WELL AS THEIR CORRESPONDING PATHINF FILES, INDEXED WITH AN ID

52:

53: player = None #Create a blank variable called player to allow player to referenced by functions as python does not support hoisting

54: clock = pygame.time.Clock() #Sets up the pygame clock to allow a framerate

55: clock.tick(60) #Sets that framerate to 60

56:

57: camera\_x = 0

58: camera\_y = 0

59:

60: sprite\_ids = [] #Initializes a list to store otherplayers sprites

61: otherplayers = pygame.sprite.Group() #Pygame group to store instances of other players

62: enemies = [] #Initializes a list for enemy IDS

63: enemysprites = pygame.sprite.Group() #Pygame group to store enemy instances

64: projectiles = pygame.sprite.Group() #Pygame group to store projectile instances

65: NPCS = pygame.sprite.Group() #Pygame group to store NPC instances

66: lockedcontainers = []

67:

68: container\_inv = {

69: "Weapons": {},

70: "Armor": {},

71: "Ammo": {},

72: "Consumables":{},

73: "Misc": {}

74: }# Template for container inventory to be referenced as a work around for hoisting once again

75:

76: interrupt = False

77: class Menu:

78: def \_\_init\_\_(self, screen):

79: self.screen = screen

80: self.running = True # Control whether the menu is active

81: # Define initial menu properties

82: self.text\_lines = ["Welcome to Wild Wasteland",

83: "Essentially Your Goal Is To Explore The Wasteland And Find A Way To Safety", "",

84: "KeyBinds:", "ESC: Close Menus", "LMB: Attack", "E: Interact", "Tab: Open Inventory",

85: "WASD / Arrow Keys: Movement In Corresponding Direction",

86: "F : Use Equipped Offhand Item", "F1: Open This Menu", "", "", "Tips:",

87: "Search Containers For Useful Items Like Weapons And Armor",

88: "Equip Said Items For Damage And Armor Buffs", "Use Ammo Sparingly", "Stock Up On Meds",

89: "Speak To The Locals "]

90: self.button\_text = "Close" # Text for the button

91: self.x = 100

92: self.line\_height = 30

93: self.button\_height = 50 # Height of the button

94: self.text\_color = (255, 255, 255)

95: self.block\_color = (128, 128, 128)

96:

97: self.font = pygame.font.Font(None, 36)

98: self.width = max(self.font.size(line)[0] for line in self.text\_lines)

99: self.height = len(self.text\_lines) \* self.line\_height + self.button\_height

100:

101: self.block = pygame.Surface((self.width, self.height))

102: self.block.fill(self.block\_color)

103:

104: # Position the button at the center bottom

105: self.button\_x = (constants.screen\_width - self.width) // 2

106: self.button\_y = constants.screen\_height - self.button\_height

107:

108: def draw(self):

109: y\_offset = self.x

110: for line in self.text\_lines:

111: text\_surface = self.font.render(line, True, self.text\_color)

112: text\_rect = text\_surface.get\_rect(topleft=(self.x, y\_offset))

113: self.screen.blit(text\_surface, text\_rect)

114: y\_offset += self.line\_height

115:

116: # Create the button at the center bottom

117: button\_surface = pygame.Surface((self.width, self.button\_height))

118: button\_surface.fill((255, 0, 0)) # Red button background

119: button\_text\_surface = self.font.render(self.button\_text, True, self.text\_color)

120: button\_text\_rect = button\_text\_surface.get\_rect(center=(self.width / 2, self.button\_height / 2))

121: self.screen.blit(button\_surface, (self.button\_x, self.button\_y))

122: self.screen.blit(button\_text\_surface, (self.button\_x + (self.width - button\_text\_rect.width) / 2, self.button\_y + (self.button\_height - button\_text\_rect.height) / 2))

123:

124: def run(self):

125: while self.running:

126: for event in pygame.event.get():

127: if event.type == pygame.QUIT:

128: self.running = False

129: elif event.type == pygame.KEYDOWN:

130: if event.key == pygame.K\_ESCAPE:

131: self.running = False

132: elif event.type == pygame.MOUSEBUTTONDOWN:

133: # Check if the mouse click is within the button's area

134: mouse\_x, mouse\_y = pygame.mouse.get\_pos()

135: button\_rect = pygame.Rect(self.button\_x, self.button\_y, self.width, self.button\_height)

136: if button\_rect.collidepoint(mouse\_x, mouse\_y):

137: self.running = False

138:

139: self.screen.fill((0, 0, 0)) # Clear the screen

140: self.draw()

141: pygame.display.flip()

142: class DeathMenu(Menu):

143: def \_\_init\_\_(self, screen, killer):

144: super().\_\_init\_\_(screen)

145: self.killer = killer

146: self.button\_text = "Close"

147: self.message = f"You were killed by {self.killer}"

148: self.message\_font = pygame.font.Font(None, 48) # You can adjust the font size

149:

150: def draw(self):

151: self.text\_lines = []

152: super().draw() # Call the parent class's draw method

153:

154: # Draw the "You were killed by (killer)" message

155: message\_surface = self.message\_font.render(self.message, True, self.text\_color)

156: message\_rect = message\_surface.get\_rect(center=(self.width / 2, self.height / 2))

157: self.screen.blit(message\_surface, message\_rect)

158:

159: def run(self):

160: start\_time = pygame.time.get\_ticks() # Get the current time

161: while self.running:

162: current\_time = pygame.time.get\_ticks()

163: if current\_time - start\_time >= 3000: # Check if 3 seconds have passed

164: self.running = False

165:

166: for event in pygame.event.get():

167: if event.type == pygame.QUIT:

168: self.running = False

169: elif event.type == pygame.KEYDOWN:

170: if event.key == pygame.K\_ESCAPE:

171: self.running = False

172:

173: self.screen.fill((0, 0, 0)) # Clear the screen

174: self.draw()

175: pygame.display.flip()

176: class OTHERPLAYER(pygame.sprite.Sprite):

177: def \_\_init\_\_(self, sid, x, y,img):

178: """

179: :param sid:

180: :param x:

181: :param y:

182: :param img:

183: """

184: super().\_\_init\_\_()

185: self.x = x

186: self.y = y

187: self.sid = sid

188: self.rect = pygame.Rect(self.x, self.y, 32, 32) # Define a rect to draw the image to

189: self.image = img

190: self.image = pygame.image.load(constants.assetpath +self.image) #Image for other player

191:

192: def update(self,x,y): #Draws the otherplayers to the map surface

193: """

194: :param x:

195: :param y:

196: :return:

197: """

198: self.x = x

199: self.y = y

200: self.rect = pygame.Rect(self.x, self.y, 32, 32)

201: otherplayers.draw(map.map\_surface)

202:

203: otherplayers.add(OTHERPLAYER(-1,-1,-1,"R.png"))

204: #Create a reference of the other players instance to allow iteration through the sprite group to draw them

205: class Map:

206: def \_\_init\_\_(self, tmx\_file, path\_file=None):

207: '''

208: Here is the map class this stores all the map related data such as objects to collide with, intercatables

209: The map class also converts the csv into a matrice for pathfinding and tmx file grids

210: :param tmx\_file:

211: :param path\_file:

212: '''

213: self.tmx\_file = constants.mappath + tmx\_file

214: self.path\_file = constants.mappath + path\_file

215: self.tmx\_data = pytmx.load\_pygame(self.tmx\_file, pixelalpha=True)

216: self.obstacles = get\_object\_rects(self.tmx\_file)

217: self.interactables = get\_interactable\_rects(self.tmx\_file)

218: self.map\_width = self.tmx\_data.width \* self.tmx\_data.tilewidth

219: self.map\_height = self.tmx\_data.height \* self.tmx\_data.tileheight

220: self.map\_surface = pygame.Surface((self.map\_width, self.map\_height))

221: self.map\_id = None

222: self.path\_data = []

223: self.collision\_boxes = []

224:

225: # load path data from csv file if provided, otherwise from tmx file

226: if path\_file:

227: with open(constants.mappath+ path\_file, 'r') as f:

228: reader = csv.reader(f)

229: for row in reader:

230: self.path\_data.append(row)

231:

232: else:

233: for layer in self.tmx\_data.layers:

234: if layer.name == 'PATHING':

235: self.path\_data = layer.data

236:

237: #Currently unused function but allows to check for the value of the current tile the player is on for interactable events like enemy spawning

238: def get\_tile\_value(self, x, y):

239: row = self.path\_data[y]

240: tile\_value = row[x]

241: return tile\_value

242:

243: #The change map fucntion is called whenever the player chnages map and upates the Map ID and hence forth updates all the new collisions and interactables on that layer of the map specifically.

244: def changemap(self,id):

245: global map\_id

246: self.map\_width = self.tmx\_data.width \* self.tmx\_data.tilewidth

247: self.map\_height = self.tmx\_data.height \* self.tmx\_data.tileheight

248: self.map\_surface = pygame.Surface((self.map\_width, self.map\_height))

249: self.obstacles = get\_object\_rects(self.tmx\_file)

250: self.interactables = get\_interactable\_rects(self.tmx\_file)

251: self.map\_id = id

252: for npc in NPCS:

253: npc.check\_map\_change(map\_id)

254: #THE NPC class is very barebones and stores the Name of the NPC as well as its image and lcoation for it to be drawn to the map

255: #The npc consists of a two step interaction couple of events first it checks if its colliding with the player if so allows the interact fucntion to be called

256: #This function will check the NPCS name and if any quests related to that npc are available creates dialogue dependant on that quest.

257: class NPC(pygame.sprite.Sprite):

258: def \_\_init\_\_(self, image\_path, map\_id, rect\_x, rect\_y,name):

259: '''

260: Initializes the NPC as a png image drawn in the rect

261: :param image\_path:

262: :param map\_id:

263: :param rect\_x:

264: :param rect\_y:

265: :param name:

266: '''

267: super().\_\_init\_\_()

268: self.image = pygame.image.load(image\_path)

269: self.map\_id = map\_id

270: self.rect = self.image.get\_rect()

271: self.rect.x = rect\_x

272: self.rect.y = rect\_y

273: self.name = name

274: self.talking = False

275: NPCS.add(self)

276:

277: def check\_collision(self, player,keys):

278: if str(self.map\_id) == str(map.map\_id):

279: if self.rect.colliderect(player.rect):

280: screentext.changetext("PRESS E TO SPEAK")

281: if keys[pygame.K\_e] or self.talking == True:

282: self.interact(player)

283: self.talking = True

284: else:

285: self.talking = False

286:

287: def interact(self,player):

288: # Represents a conversation between the player and the NPC, its behavior varies by NPC's name, the player's current quests, and inventory.

289: if self.name == "Tony":

290: if any(element == (0,0) for element in player.quests):

291: screentext.changetext("HI MY NAMES TONY CAN YOU FIND MY KEY?")

292: if 'Key' in player.inventory['Misc']:

293: QUESTUPDATE["ID"] = "0"

294: QUESTUPDATE["VALUE"] = "1"

295: s.send((json.dumps(QUESTUPDATE) + "#").encode())

296: screentext.changetext("THANK YOU FOR FINDING MY KEY HAVE THIS")

297: player.inventory["Misc"]["Key"] -= 1

298: player.quests.remove((0,0))

299: self.talking = False

300: else:

301: screentext.changetext("I DONT NEED YOUR HELP RIGHT NOW")

302: if self.name == "Shade":

303: if any(element == (999, 0) for element in player.quests):

304: if "Cap(s)" in player.inventory.get("Misc"):

305: if player.inventory["Misc"]["Cap(s)"] >= 10000:

306: screentext.changetext("Now Were Talking")

307: RMV["AMMO"] = "Cap(s)"

308: RMV["AMOUNT"] = 10000

309: s.send((json.dumps(RMV) + "#").encode())

310: player.inventory["Misc"]["Cap(s)"] -= 10000

311: QUESTUPDATE["ID"] = "999"

312: QUESTUPDATE["VALUE"] = "1"

313: s.send((json.dumps(QUESTUPDATE) + "#").encode())

314: else:

315: screentext.changetext("Come Back When You Have Some Cash!")

316: self.talking = False

317: if self.name == "Wrangler":

318: screentext.changetext("Thanks for saving my skin")

319: if any(element == (1, 0) for element in player.quests):

320: screentext.changetext("Would You Mind Finding Me A Machete")

321: if any(element == (1, 0) for element in player.quests) and "Machete" in player.inventory['Weapons']:

322: screentext.changetext("Hey I Was Just Looking For A Machete")

323: QUESTUPDATE["ID"] = "1"

324: QUESTUPDATE["VALUE"] = "1"

325: s.send((json.dumps(QUESTUPDATE) + "#").encode())

326: del player.inventory["Weapons"]["Machete"]

327: screentext.changetext("Here Take This Old Tool I Had")

328: self.talking = False

329:

330:

331:

332:

333:

334:

335:

336: #The check map change is a function that is called to check the current map

337: #If the map is not the same as the npcs map id it is removed and killed to prevent NPCs spawning on the wrong map layer

338: def check\_map\_change(self, map\_id):

339: if map\_id != self.map\_id:

340: NPCS.remove(self)

341: self.kill()

342: #Here is the player class this is the main instance the user controls when using the game

343: class Player(pygame.sprite.Sprite):

344: global screentext

345: global fpressed

346: #Screen text is a global variable to allow instant reference to it dependant on the players actions

347:

348: def \_\_init\_\_(self, x, y, map):

349: super().\_\_init\_\_()

350: #ANIMATIONS//CONSTANTS used to create the Character

351: self.name = "p"

352: self.map = map

353: self.x = x

354: self.y = y

355: self.max\_hp = 100

356: self.hp = 100

357: self.speed = 5

358: self.direction = "left"

359:

360: self.counter = 0 #Counter to identify which step of the animation loop the player is in

361: self.walking\_index = 0

362: #Animations from PNG

363: self.walkleft = [pygame.image.load(constants.assetpath+ "left4.png"),

364: pygame.image.load(constants.assetpath+"left2.png"),

365: pygame.image.load(constants.assetpath+"left3.png"),

366: pygame.image.load(constants.assetpath+"left.png")] #Dictionaries storing varous animation loops

367: self.walkright = [pygame.image.load(constants.assetpath+"right4.png"),

368: pygame.image.load(constants.assetpath+"right2.png"),

369: pygame.image.load(constants.assetpath+"right3.png"),

370: pygame.image.load(constants.assetpath+"right.png")]

371: self.dying = [pygame.image.load(constants.assetpath+"death\_1.png"),

372: pygame.image.load(constants.assetpath+"death\_2.png"),

373: pygame.image.load(constants.assetpath+"death\_3.png")]

374: self.image = self.walkleft[0]

375:

376:

377: self.rect = pygame.Rect(self.x, self.y, 32, 64)

378: self.lastdmgtype = "Gun" #This is the damage type which is displayed when you die telling you chat killed you

379: self.lastshot = time.time()

380: self.damage\_reduc = 1

381: #The equipped weapon type changes between guns and melee to indicate hwo the player should attack

382:

383: self.inventory = {

384: "Weapons": {},

385: "Armor": {},

386: "Ammo": {},

387: "Consumables": {},

388: "Misc": {}

389: } #Template for the players inventory

390: self.equipped = {

391: "H":{},

392: "C": {},

393: "F" :{},

394: "Hand": {},

395: "Offhand": {}

396: }

397: self.continv = {

398: "Weapons": {},

399: "Armor": {},

400: "Ammo": {},

401: "Consumables":{},

402: "Misc": {}

403: } # Template for conatiner inventory

404: self.quests = set() # Set used to store quests to ensure only one instance of each quest exists

405: self.contID = "" # The containerID currently being accessed

406: self.equipped\_images = {}

407: self.lockedcontainers = []

408:

409: self.lastshotupdate = time.time()

410: self.shots = 0

411: self.lastammo = ''

412:

413: def statupdate(self):

414: temparmor = 1

415: for i in player.equipped:

416: if i in ["H","C","F"]:

417: try:

418: armor\_data = items.Armor.get(player.equipped[i])

419: if armor\_data:

420: temparmor += armor\_data[2] # Index 2 for the armor value

421: except:

422: pass

423:

424: player.damage\_reduc = temparmor

425:

426: def update(self, keys):

427: global fpressed

428: global camera\_x

429: global camera\_y

430: '''

431: The main method of the player class the update function is a recursive function called in the main loop to allow

432: movement as well as handling all of the keybinds, the update fucntion also is the main driver in collision handling

433: it cross refernces the map Classes object list against th players rect to determine collision

434: It also handles intercatable types such as doors or containers.

435: :param keys:

436: :return:

437: '''

438: self.enlarged\_rect = self.rect.inflate(10, 10)

439: global time\_factor

440: dx, dy = 0, 0

441:

442: if self.hp <= 0:

443: #Script to handle death by cheking health and reverts to first map upon death

444: print("You died to a ", self.lastdmgtype)

445: self.x = 0

446: self.y = 0

447: changemaps("0")

448: self.rect.x, self.rect.y = 0, 0

449: self.hp = 100

450: d = DeathMenu(screen, self.lastdmgtype)

451: d.run()

452:

453: if pygame.mouse.get\_pressed()[0]:

454: #uses mouse position to get target location and creates an instance of a projectile

455: # using the equipped item type and relevant ammo

456: mouse\_x, mouse\_y = pygame.mouse.get\_pos()

457:

458:

459: if self.equipped["Hand"] != {}:

460: weapondata = items.Weapons.get(self.equipped["Hand"])

461: if time.time() - self.lastshot > weapondata[2]:

462: self.lastshot = time.time()

463: Projectile(self.equipped["Hand"],self,(mouse\_x -camera\_x, mouse\_y-camera\_y))

464:

465:

466: if time.time() - self.lastshotupdate >= 2:

467: if self.lastammo != '' and self.shots != 0:

468: RMV["AMMO"] = self.lastammo

469: RMV["AMOUNT"] = self.shots

470: s.send((json.dumps(RMV) + "#").encode())

471: self.shots = 0

472: self.lastshotupdate = time.time()

473:

474:

475: if keys[pygame.K\_UP] or keys[pygame.K\_w]:

476: dy = -self.speed

477: self.direction = "up"

478: self.counter += 1

479:

480: if keys[pygame.K\_TAB]:

481: inventory\_menu = InventoryMenu(constants.screen\_width, constants.screen\_height, self)

482: inventory\_menu.show\_inventory\_menu() #INVENTORY

483:

484: if keys[pygame.K\_SPACE]:

485: self.rect.x = 0

486: self.rect.y = 0

487:

488: if keys[pygame.K\_F1]:

489: menu = Menu(screen)

490: menu.run()

491:

492: if keys[pygame.K\_f] and not fpressed:

493: fpressed = True

494: if self.equipped["Offhand"] != {}:

495: consumabledata = items.Consumables.get(self.equipped["Offhand"])

496: if consumabledata[1] == "HEAL":

497: player.hp += consumabledata[2]

498: if player.hp > 100:

499: player.hp = 100

500: self.inventory["Consumables"][self.equipped["Offhand"]] -= 1

501: RMV["AMMO"] = self.equipped["Offhand"]

502: RMV["AMOUNT"] = 1

503: s.send((json.dumps(RMV) + "#").encode())

504: if self.inventory["Consumables"][self.equipped["Offhand"]] <= 0:

505: del self.inventory["Consumables"][self.equipped["Offhand"]]

506: self.equipped["Offhand"] = {}

507: with open("equipped\_items.json", "r") as json\_file:

508: equipped\_data = json.load(json\_file)

509: equipped\_data["Offhand"] = ""

510: with open("equipped\_items.json", "w") as json\_file:

511: json.dump(equipped\_data, json\_file)

512:

513: if keys[pygame.K\_DOWN] or keys[pygame.K\_s]:

514: dy = self.speed

515: self.direction = "down"

516: self.counter += 1

517:

518: if keys[pygame.K\_LEFT] or keys[pygame.K\_a]:

519: dx = -self.speed

520: self.direction = "left"

521: self.counter += 1

522:

523: if keys[pygame.K\_RIGHT] or keys[pygame.K\_d]:

524: dx = self.speed

525: self.direction = "right"

526: self.direction = "right"

527: self.counter += 1

528:

529: if keys[pygame.K\_LEFT] or keys[pygame.K\_RIGHT] or keys[pygame.K\_UP] or keys[pygame.K\_DOWN] or keys[pygame.K\_a] or keys[pygame.K\_s] or keys[pygame.K\_d] or keys[pygame.K\_w]:

530: if self.counter >= 10:

531: self.walking\_index = (self.walking\_index + 1) % len(self.walkleft)

532: self.counter = 0

533:

534: #Manages hovering over the player to display their health bar

535: mouse\_x , mouse\_y = pygame.mouse.get\_pos()

536: if self.rect.collidepoint(((mouse\_x - camera\_x),(mouse\_y-camera\_y))):

537: self.draw\_overhead\_health\_bar(map.map\_surface)

538: # Check for collisions with obstacles

539: next\_rect = self.rect.move(dx, dy)

540: for obstacle in map.obstacles:

541: if obstacle.colliderect(next\_rect):

542: return

543:

544: #manages intercatbles

545: for inter in map.interactables:

546: if self.enlarged\_rect.colliderect(inter[1]):

547: if inter[0] == 'cont':

548: screentext.changetext("PRESS E TO OPEN")

549: if keys[pygame.K\_e]:

550: self.contID = inter[2]

551: openCont(inter[2])

552: elif inter[0] == 'door':

553: screentext.changetext("PRESS E TO ENTER")

554: if keys[pygame.K\_e]:

555: changemaps(str(inter[2]))

556:

557: # Update position if no collisions

558: self.rect.move\_ip(dx, dy)

559: self.rect.clamp\_ip(pygame.Rect(0, 0, map.map\_width, map.map\_height))

560:

561: if self.direction == "left":

562: self.image = self.walkleft[self.walking\_index]

563: else:

564: self.image = self.walkright[self.walking\_index]

565: def draw\_overhead\_health\_bar(self, screen):

566: '''

567: Simple ratio method to handle the displaying of the over head health bar when hovering over your player

568: :param screen:

569: :return:

570: '''

571: # Define colors

572: HEALTH\_BAR\_BG\_COLOR = constants.RED # Red background color

573: HEALTH\_BAR\_COLOR = constants.GREEN # Green health color

574: BORDER\_COLOR = constants.BLACK # Black border color

575:

576: # Calculate the width and height of the health bar

577: bar\_width = self.rect.width

578: bar\_height = 5

579:

580: # Calculate the current width of the health bar based on the current HP and max HP

581: current\_health\_width = int((self.hp / self.max\_hp) \* bar\_width)

582:

583: # Calculate the position of the health bar above the enemy

584: health\_bar\_x = self.rect.x

585: health\_bar\_y = self.rect.y - bar\_height - 2

586:

587: # Draw the background of the health bar

588: pygame.draw.rect(screen, HEALTH\_BAR\_BG\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height))

589:

590: # Draw the current health portion of the health bar

591: pygame.draw.rect(screen, HEALTH\_BAR\_COLOR, (health\_bar\_x, health\_bar\_y, current\_health\_width, bar\_height))

592:

593: # Draw the border of the health bar

594: pygame.draw.rect(screen, BORDER\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height), 1)

595: class InteractableText():

596: def \_\_init\_\_(self, font\_size, text, color, screen):

597: pygame.font.init()

598: self.font = pygame.font.Font(None, font\_size)

599: self.screen = screen

600: self.color = color

601:

602: # Load the image and scale it to be 3 times smaller

603: self.image = pygame.image.load(constants.assetpath + "notification.png").convert()

604: original\_size = self.image.get\_size()

605: new\_size = (original\_size[0] // 3, original\_size[1] // 3)

606: self.image = pygame.transform.scale(self.image, new\_size)

607:

608: self.changetext("")

609: self.questnotification = False

610: self.questtime = 0

611: self.text\_rect = self.text\_surface.get\_rect(center=((screen\_width // 2) -200 , (screen\_height - self.font.get\_height() // 2) - 50))

612: self.quest\_text\_rect = self.text\_surface.get\_rect(center=((screen\_width // 2)- 200, (screen\_height - self.font.get\_height() // 2)))

613:

614: def changetext(self, text):

615: self.text = text

616: self.text\_surface = self.font.render(text, True, self.color)

617:

618: def quest(self, reward, amount, sender):

619: self.questtime = time.time()

620: self.questnotification = True

621: self.questtime = time.time()

622: self.quest\_text = f'You Received {amount} {reward} from {sender}'

623: self.quest\_text\_surface = self.font.render(self.quest\_text, True, constants.WHITE)

624:

625: def draw(self, time):

626: self.screen.blit(self.text\_surface, self.text\_rect)

627: if time - self.questtime >= 3:

628: self.questtime = time

629: self.questnotification = False

630: if self.questnotification == True:

631: self.screen.blit(self.image, (screen\_width - self.image.get\_width(), screen\_height -self.image.get\_height()))

632: self.screen.blit(self.quest\_text\_surface, self.quest\_text\_rect)

633:

634: screentext = InteractableText(48, "", (25,149,21), screen)

635: #Initiallize an instance of screentext for workaround to hoisting

636: class Enemy(pygame.sprite.Sprite):

637: global enemies

638: global enemysprites

639: def \_\_init\_\_(self, ID, rectx, recty, speed, name, health, image,damage, map,cooldown):

640: '''

641: The enemy class is slightly different as a single instance is never created instead the class method

642: from csv is used to build the class from information proviided by the server similarly the fromcsv class

643: also handles movement, however all basic attributes are found below.

644: :param ID:

645: :param rectx:

646: :param recty:

647: :param speed:

648: :param name:

649: :param health:

650: :param image:

651: :param map:

652: '''

653: super().\_\_init\_\_()

654: self.ID = ID

655: self.x = rectx // 32

656: self.y = recty // 32

657: self.name = name

658: self.health = health

659: self.image = image

660: self.rect = pygame.Rect(rectx // 32, recty // 32, 32, 32)

661: self.rect.x = rectx

662: self.rect.y = recty

663: self.max\_health = 100

664: self.hp = hp

665: self.map = map

666: self.damage = int(damage)

667: self.speed = int(speed)

668: self.alive = True

669: self.player = player

670: self.targetcoord = None # Initialize target coordinate as None

671: enemysprites.add(self)

672: enemies.append(self.ID)

673: self.cooldown = float(cooldown) # Cooldown duration in seconds

674: self.last\_attack\_time = 0

675: self.equippedweapon = "Melee"

676:

677: def update(self):

678: '''

679: This checks if the enemy is hit and tells the server it has been hit by said player

680: Futhermore the enemy will also check if its been hit and has a target if not whoever hit it will be immediatley tracked

681: and the enemy will pathfind towards said player.

682: :return:

683: '''

684: for proj in projectiles:

685: if self.rect.colliderect(proj) and proj.map == self.map:

686: self.hit(proj.damage)

687:

688: # Check if target coordinate is set and move towards it

689: if self.targetcoord is not None:

690: target\_x, target\_y = self.targetcoord

691: dx = target\_x - self.rect.x

692: dy = target\_y - self.rect.y

693: distance = math.sqrt(dx \*\* 2 + dy \*\* 2)

694: if distance >= 0:

695: try:

696: vx = dx / distance

697: vy = dy / distance

698: self.rect.x += vx \* self.speed

699: self.rect.y += vy \* self.speed

700: except ZeroDivisionError:

701: pass

702:

703: def hit(self, dmg):

704: ENEMYHIT["ID"] = self.ID

705: ENEMYHIT["DMG"] = dmg

706: s.send((json.dumps(ENEMYHIT) + "#").encode())

707:

708: self.hp -= dmg

709:

710: if self.hp <= 0:

711: self.alive = False

712: self.kill()

713: enemysprites.remove(self)

714:

715:

716: self.damage\_text = f"-{dmg}" #formatted the damage text as the dmg taken

717: self.damage\_timer = pygame.time.get\_ticks() + 1000 #display for 1 second

718:

719: def draw\_health\_bar(self, screen):

720: '''

721: Method copied from the player class to display health of the enemy

722: However this method is recusrivley called by the main loop so the enemies health is always shown

723: :param screen:

724: :return:

725: '''

726: if str(self.map) == str(map.map\_id):

727: HEALTH\_BAR\_BG\_COLOR = constants.RED # Red background color

728: HEALTH\_BAR\_COLOR = constants.GREEN # Green health color

729: BORDER\_COLOR = constants.BLACK # Black border color

730:

731: # Calculate the width and height of the health bar

732: bar\_width = self.rect.width

733: bar\_height = 5

734:

735: # Calculate the current width of the health bar based on the current HP and max HP

736: current\_health\_width = int((self.hp / self.max\_health) \* bar\_width)

737:

738: # Calculate the position of the health bar above the enemy

739: health\_bar\_x = self.rect.x

740: health\_bar\_y = self.rect.y - bar\_height - 2

741:

742: # Draw the background of the health bar

743: pygame.draw.rect(screen, HEALTH\_BAR\_BG\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height))

744:

745: # Draw the current health portion of the health bar

746: pygame.draw.rect(screen, HEALTH\_BAR\_COLOR, (health\_bar\_x, health\_bar\_y, current\_health\_width, bar\_height))

747:

748: # Draw the border of the health bar

749: pygame.draw.rect(screen, BORDER\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height), 1)

750:

751: if hasattr(self, 'damage\_text') and pygame.time.get\_ticks() < self.damage\_timer:

752: damage\_font = pygame.font.Font(None, 36)

753: damage\_text = damage\_font.render(self.damage\_text, True, (255, 0, 0)) # Red text

754: damage\_text\_rect = damage\_text.get\_rect()

755: damage\_text\_rect.center = (self.rect.centerx, self.rect.y - 20) # Adjust the position

756:

757: screen.blit(damage\_text, damage\_text\_rect)

758:

759: def check\_collision(self, player):

760: '''

761: THis checks if the player is in contact with the enemy and if so handles the cooldown between enemy attacks

762: allowing the enemy to attack the player.

763:

764: Ive choosen to handle enemy damage client side in an effort to avoid desync between the client and server

765: allowing the what you see is what you get approach.

766: :param player:

767: :return:

768: '''

769: current\_time = time.time()

770: if current\_time - self.last\_attack\_time >= self.cooldown and str(self.map) == str(map.map\_id):

771: if self.rect.colliderect(player.rect):

772: e = str(self.name)

773: player.lastdmgtype = e

774: player.hp -= int(self.damage \* (1- (player.damage\_reduc/100)))

775: print(int(self.damage \* (1- (player.damage\_reduc/100))))

776: # Update the last attack time

777: self.last\_attack\_time = current\_time

778: @classmethod

779: def from\_csv(cls, name, rectx, recty, ID, hp, map):

780: '''

781: This is the method that is called to actually create the enemy, the server will send a packet if the (enemy) is on your layer

782: the recv\_server method will see this gather the key info like the type of enemy, position, health and map of the enemy

783: and the instance will be created. this method is called to update enemy positioning also by simply checking if the ID

784: exists in the enemies lsit if so it will not create a new instance just update that instances values

785: :param name:

786: :param rectx:

787: :param recty:

788: :param ID:

789: :param hp:

790: :param map:

791: :return:

792: '''

793:

794: if ID not in enemies:

795: # Open the CSV file and read its contents

796: with open(constants.configpath+'enemies.csv', 'r') as file:

797: reader = csv.DictReader(file)

798: # Loop through each row in the file

799: for row in reader:

800: # Check if the name in the row matches the name of the enemy we're looking for

801: if row['name'] == name:

802: # Create a new instance of the Enemy class with the attributes from the row

803: enemy = cls(

804: ID=ID,

805: name=row['name'],

806: rectx=rectx,

807: recty=recty,

808: speed=row['speed'],

809: health=hp,

810: map=map,

811: damage = row['damage'],

812: image=pygame.image.load(constants.assetpath+ row['image']),

813: cooldown=row['cooldown']

814: )

815: return enemy

816: else:

817: if hp > 0:

818: for enemy in enemysprites.sprites():

819: if hasattr(enemy, "ID") and enemy.ID == ID:

820: enemy.targetcoord = (rectx, recty) # Set the target coordinate as a tuple

821: continue

822: if hp <= 0:

823: for enemy in enemysprites:

824: if enemy.ID == ID:

825: enemies.remove(ID)

826: enemy.alive = False

827: enemysprites.remove(enemy)

828: class Transfer:

829: global interrupt

830: invimgs = []

831: def \_\_init\_\_(self, player):

832: '''

833: This is the constructor for the transfer window that is displayed when interacting with a container

834: the paramater player is used so the inventory is able to be referenced allowing updates to it

835: :param player:

836: '''

837: self.player = player

838: self.container\_inv = player.continv

839: self.selected\_tab = "Weapons" #Initalizes the transfer window to open on the weapons tab

840: self.transfer\_all = False # Flag to indicate transferring all items

841: self.screen = None

842: self.font = None

843: self.hovered\_item = None

844: self.hovered\_player\_item = None

845: self.item = None

846: self.highlight\_color = constants.BLUE

847: self.type = None

848:

849: def getitemdata(self):

850: '''

851: The item data method is called whenever an object is hovered and displays the releveant data about that item

852: from the items.py import and displays this to the user.

853: :return:

854: '''

855: if self.hovered\_item:

856: self.item = self.hovered\_item

857: if self.hovered\_player\_item:

858: self.item = self.hovered\_player\_item

859:

860: if self.item:

861: if self.selected\_tab == "Weapons":

862: item\_data = items.Weapons.get(self.item)

863: if item\_data:

864: item = self.item

865: img\_path = (item\_data[0])

866: dmg = item\_data[1]

867: firerate = item\_data[2]

868: ammo = item\_data[3]

869: img = pygame.image.load(constants.assetpath+img\_path)

870: imgrect = img.get\_rect()

871: imgrect.center = (screen\_width // 2, screen\_height - 50)

872: self.screen.blit(img, imgrect)

873: elif self.selected\_tab == "Armor":

874: item\_data = items.Armor.get(self.item)

875: if item\_data:

876: item = self.item

877: img\_path = (item\_data[0])

878: slot = item\_data[1]

879: armorval = item\_data[2]

880: img = pygame.image.load(constants.assetpath+img\_path)

881: imgrect = img.get\_rect()

882: imgrect.center = (screen\_width // 2, screen\_height - 50)

883: self.screen.blit(img, imgrect)

884: elif self.selected\_tab == "Ammo":

885: item\_data = items.Ammo.get(self.item)

886: if item\_data:

887: item = self.item

888: img\_path = (item\_data[0])

889: bulletvel = item\_data[1]

890: penval = item\_data[2]

891: img = pygame.image.load(constants.assetpath+img\_path)

892: imgrect = img.get\_rect()

893: imgrect.center = (screen\_width // 2, screen\_height - 50)

894: self.screen.blit(img, imgrect)

895: elif self.selected\_tab == "Consumables":

896: item\_data = items.Consumables.get(self.item)

897: if item\_data:

898: item = self.item

899: img\_path = (item\_data[0])

900: effect = item\_data[1]

901: value = item\_data[2]

902: img = pygame.image.load(constants.assetpath+img\_path)

903: imgrect = img.get\_rect()

904: imgrect.center = (screen\_width // 2, screen\_height - 50)

905: self.screen.blit(img, imgrect)

906: elif self.selected\_tab == "Misc":

907: item\_data = items.Misc.get(self.item)

908: if item\_data:

909: item = self.item

910: img\_path = (item\_data[0])

911: MiscID = item\_data[1]

912: img = pygame.image.load(constants.assetpath+img\_path)

913: imgrect = img.get\_rect()

914: imgrect.center = (screen\_width // 2, screen\_height - 50)

915: self.screen.blit(img, imgrect)

916:

917: def initialize(self):

918: #Initilaizes fonts and crates a window

919: self.screen = pygame.display.set\_mode((screen\_width, screen\_height))

920: self.initialize\_fonts()

921: pygame.display.set\_caption("Container"+str(player.contID))

922:

923: def initialize\_fonts(self):

924: pygame.font.init()

925: self.font = pygame.font.Font(None, constants.FONT\_SIZE)

926:

927: def draw\_inventory(self):

928: '''

929: This is the main driver in drawing all of the UI elements and calles methods to draw the releveant data to the screen

930: :return:

931: '''

932: self.screen.fill(constants.GRAY) # Set the left side of the screen to gray

933: pygame.draw.rect(self.screen, constants.DARK\_GRAY, pygame.Rect(screen\_width // 2, 0, screen\_width // 2, screen\_height)) # Set the right side of the screen to dark gray

934: self.draw\_tabs()

935: self.draw\_container\_inventory()

936: self.draw\_player\_inventory()

937: if self.hovered\_item or self.hovered\_player\_item:

938: self.getitemdata()

939: pygame.display.flip()

940:

941: def draw\_tabs(self):

942: 'Lists the tabs using the players inventory as a template'

943: tabs = list(self.player.inventory.keys())

944: tab\_pos = constants.PLAYER\_INVENTORY\_POS[0]

945: for tab in tabs:

946: tab\_rect = pygame.Rect(tab\_pos, constants.PLAYER\_INVENTORY\_POS[1] - constants.TAB\_HEIGHT, constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

947: pygame.draw.rect(self.screen, constants.BLUE if tab == self.selected\_tab else constants.GRAY, tab\_rect)

948: text = self.font.render(tab, True, constants.WHITE)

949: self.screen.blit(text, (tab\_pos + 10, constants.PLAYER\_INVENTORY\_POS[1] - constants.TAB\_HEIGHT + 5))

950: tab\_pos += constants.TAB\_WIDTH

951:

952: def draw\_player\_inventory(self):

953: player\_inv = self.player.inventory[self.selected\_tab]

954: item\_pos = list(constants.PLAYER\_INVENTORY\_POS)

955: for item, count in player\_inv.items():

956: item\_rect = pygame.Rect(item\_pos[0], item\_pos[1], constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

957: pygame.draw.rect(self.screen, constants.GRAY, item\_rect)

958: if item\_rect.collidepoint(pygame.mouse.get\_pos()):

959: self.hovered\_player\_item = item

960: if self.transfer\_all or (pygame.key.get\_mods() & pygame.KMOD\_SHIFT and item == self.hovered\_player\_item or item == self.hovered\_item):

961: self.highlight\_color = constants.RED # Change highlight color to red

962: else:

963: self.highlight\_color = constants.BLUE # Change highlight color back to blue

964: pygame.draw.rect(self.screen, self.highlight\_color, item\_rect, 3) # Add highlight border

965: self.hovered\_player\_item = item # Store the hovered item in player's inventory

966: text = self.font.render(f"{count}x {item}", True, constants.BLACK)

967: self.screen.blit(text, (item\_pos[0] + 10, item\_pos[1] + 10))

968: item\_pos[1] += constants.TAB\_HEIGHT

969:

970: def draw\_container\_inventory(self):

971: item\_pos = list(constants.CONTAINER\_INVENTORY\_POS)

972: pygame.draw.rect(self.screen, constants.DARK\_GRAY,

973: pygame.Rect(constants.CONTAINER\_INVENTORY\_POS[0], 20, constants.TAB\_WIDTH, constants.TAB\_HEIGHT)) # Container header

974: text = self.font.render("Container", True, constants.WHITE)

975: self.screen.blit(text, (constants.CONTAINER\_INVENTORY\_POS[0] + 10, 25))

976: container\_items = []

977: for item, count in self.container\_inv[self.selected\_tab].items():

978: container\_items.append((item, count)) # Store items in a separate list

979: for item, count in container\_items:

980: item\_rect = pygame.Rect(item\_pos[0], item\_pos[1], constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

981: pygame.draw.rect(self.screen, constants.DARK\_GRAY, item\_rect)

982: if item\_rect.collidepoint(pygame.mouse.get\_pos()):

983: if self.transfer\_all or (pygame.key.get\_mods() & pygame.KMOD\_SHIFT and item == self.hovered\_item):

984: self.highlight\_color = constants.RED # Change highlight color to red

985: else:

986: self.highlight\_color = constants.BLUE # Change highlight color back to blue

987: pygame.draw.rect(self.screen, self.highlight\_color, item\_rect, 3) # Add highlight border

988: self.hovered\_item = item # Store the hovered item

989: text = self.font.render(f"{count}x {item}", True, constants.BLACK)

990: self.screen.blit(text, (item\_pos[0] + 10, item\_pos[1] + 10))

991: item\_pos[1] += constants.TAB\_HEIGHT

992:

993: def handle\_events(self):

994: '''

995: This handles keybinds in the transfer for events such as shift clicking items which results in the whole

996: stack being transfered

997: :return:

998: '''

999: for event in pygame.event.get():

1000: if event.type == pygame.QUIT:

1001: pygame.quit()

1002: sys.exit()

1003: elif event.type == pygame.KEYDOWN:

1004: if event.key == pygame.K\_ESCAPE:

1005: player.continv = {

1006: "Weapons": {},

1007: "Armor": {},

1008: "Ammo": {"9mm Ammo": 1},

1009: "Consumables": {},

1010: "Misc": {}

1011: }

1012: self.invrunning = False

1013: elif event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

1014: self.handle\_click(event.pos)

1015: elif event.type == pygame.KEYDOWN:

1016: if event.key == pygame.K\_LSHIFT:

1017: self.transfer\_all = True

1018: elif event.type == pygame.KEYUP:

1019: if event.key == pygame.K\_LSHIFT:

1020: self.transfer\_all = False

1021:

1022: def handle\_click(self, pos):

1023: # Check if tab is clicked

1024: tabs = list(self.player.inventory.keys())

1025: tab\_pos = constants.PLAYER\_INVENTORY\_POS[0]

1026: for tab in tabs:

1027: tab\_rect = pygame.Rect(tab\_pos, constants.PLAYER\_INVENTORY\_POS[1] - constants.TAB\_HEIGHT, constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

1028: if tab\_rect.collidepoint(pos) and self.selected\_tab != tab:

1029: self.selected\_tab = tab

1030: self.selected\_item = None # Reset the selected item when switching tabs

1031: break

1032: tab\_pos += constants.TAB\_WIDTH

1033: self.hovered\_player\_item = None

1034:

1035: # Check if item is clicked in player inventory

1036: player\_inv = self.player.inventory[self.selected\_tab]

1037: item\_pos = list(constants.PLAYER\_INVENTORY\_POS)

1038: for item, count in player\_inv.items():

1039: item\_rect = pygame.Rect(item\_pos[0], item\_pos[1], constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

1040: if item\_rect.collidepoint(pos):

1041: self.type = TOCONT

1042: if self.transfer\_all:

1043: self.transfer\_all\_items(player\_inv, self.container\_inv[self.selected\_tab], item)

1044: elif pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

1045: self.transfer\_all\_items(player\_inv, self.container\_inv[self.selected\_tab], item)

1046: else:

1047: self.transfer\_item(player\_inv, self.container\_inv[self.selected\_tab], item)

1048: self.hovered\_item = None # Clear the hovered item

1049: break

1050: item\_pos[1] += constants.TAB\_HEIGHT

1051:

1052: # Check if item is clicked in container inventory

1053: container\_inv = self.container\_inv[self.selected\_tab]

1054: item\_pos = list(constants.CONTAINER\_INVENTORY\_POS)

1055: for item, count in container\_inv.items():

1056: item\_rect = pygame.Rect(item\_pos[0], item\_pos[1], constants.TAB\_WIDTH, constants.TAB\_HEIGHT)

1057: if item\_rect.collidepoint(pos):

1058: self.type = FROMCONT

1059: if self.transfer\_all:

1060: self.transfer\_all\_items(container\_inv, self.player.inventory[self.selected\_tab], item)

1061: elif pygame.key.get\_mods() & pygame.KMOD\_SHIFT:

1062: self.transfer\_all\_items(container\_inv, self.player.inventory[self.selected\_tab], item)

1063: else:

1064: self.transfer\_item(container\_inv, self.player.inventory[self.selected\_tab], item)

1065: self.hovered\_item = None # Clear the hovered item

1066: break

1067: item\_pos[1] += constants.TAB\_HEIGHT

1068:

1069: def transfer\_item(self, from\_inventory, to\_inventory, item):

1070: '''

1071: Called when you click an item moves from the one inventory to the other

1072: :param from\_inventory:

1073: :param to\_inventory:

1074: :param item:

1075: :return:

1076: '''

1077:

1078: self.type["TAB"] = self.selected\_tab

1079: self.type["Item"] = item

1080: self.type["A"] = 1

1081: self.type["ID"] = player.contID

1082: from\_inventory[item] -= 1

1083: if from\_inventory[item] == 0:

1084: del from\_inventory[item]

1085: s.send((json.dumps(self.type) + "#").encode())

1086: to\_inventory[item] = to\_inventory.get(item, 0) + 1

1087:

1088: if self.selected\_tab == "Armor":

1089: if any(item in player.equipped[slot] for slot in ["H", "C", "F"]) and item not in player.inventory.get("Armor"):

1090: for slot in ["H", "C", "F"]:

1091: if item in player.equipped[slot]:

1092: player.equipped[slot] = {}

1093: player.statupdate()

1094: if self.selected\_tab == "Weapons":

1095: if item in player.equipped["Hand"] and item not in player.inventory.get("Weapons"):

1096: if item in player.equipped["Hand"]:

1097: player.equipped["Hand"] = {}

1098: if self.selected\_tab == "Consumables":

1099: if item in player.equipped["Offhand"] and item not in player.inventory.get("Consumables"):

1100: if item in player.equipped["Offhand"]:

1101: player.equipped["Offhand"] = {}

1102:

1103:

1104:

1105:

1106: def transfer\_all\_items(self, from\_inventory, to\_inventory, item=None):

1107: if item is None:

1108: for item in list(from\_inventory.keys()):

1109: count = from\_inventory[item]

1110: to\_inventory[item] = to\_inventory.get(item, 0) + count

1111: del from\_inventory[item]

1112: else:

1113: count = from\_inventory[item]

1114: self.type["TAB"] = self.selected\_tab

1115: self.type["Item"] = item

1116: self.type["A"] = count

1117: self.type["ID"] = player.contID

1118: s.send((json.dumps(self.type) + "#").encode())

1119: to\_inventory[item] = to\_inventory.get(item, 0) + count

1120: del from\_inventory[item]

1121: return # Exit the method after transferring the specified item

1122:

1123: if self.selected\_tab == "Armor":

1124: if any(item in player.equipped[slot] for slot in ["H", "C", "F"]) and item not in player.inventory.get("Armor"):

1125: for slot in ["H", "C", "F"]:

1126: if item in player.equipped[slot]:

1127: player.equipped[slot] = {}

1128: player.statupdate()

1129: if self.selected\_tab == "Weapons":

1130: if item in player.equipped["Hand"] and item not in player.inventory.get("Weapons"):

1131: if item in player.equipped["Hand"]:

1132: player.equipped["Hand"] = {}

1133: if self.selected\_tab == "Consumables":

1134: if item in player.equipped["Offhand"] and item not in player.inventory.get("Consumables"):

1135: if item in player.equipped["Offhand"]:

1136: player.equipped["Offhand"] = {}

1137: def run(self):

1138: '''

1139: The main run loop which recursivley calls fucntions as well as cheking for enemy collisons while in a container

1140: as opening a container would prevent enemy damage otherwise

1141: :return:

1142: '''

1143: self.initialize()

1144: self.invrunning = True

1145: while self.invrunning and interrupt == False:

1146: for enemy in enemysprites.sprites():

1147: enemy.check\_collision(player)

1148: if player.hp <= 0:

1149: self.invrunning = False

1150: clock.tick(60)

1151: self.handle\_events()

1152: self.draw\_inventory()

1153: self.hovered\_player\_item = None

1154: self.hovered\_item = None

1155: self.item = None

1156: class Projectile(pygame.sprite.Sprite):

1157: def \_\_init\_\_(self, weapon, source, target\_pos):

1158: super().\_\_init\_\_()

1159: self.source = source

1160: self.sourcex = source.rect.x

1161: self.sourcey = source.rect.y

1162: self.target\_pos = target\_pos

1163: self.rect = None

1164: self.alivetime = time.time()

1165: self.lastdistance = 99999

1166:

1167:

1168: # Calculate data based on the weapon

1169: if weapon != {}:

1170: weapon\_data = items.Weapons.get(weapon)

1171: self.caliber = weapon\_data[3]

1172: self.damage = weapon\_data[1]

1173: bullet\_data = items.Ammo.get(self.caliber)

1174: player.lastammo = self.caliber

1175: self.speed = 100 \* bullet\_data[1]

1176: else:

1177: self.caliber = None

1178: self.damage = 1

1179: self.speed = 0

1180:

1181: self.map = source.map

1182:

1183: # Create a bullet if caliber is not None, otherwise create an inflated rect

1184: if self.caliber is not None:

1185: self.rect = pygame.Rect(source.rect.center, (5, 5))

1186: else:

1187: self.rect = source.rect.inflate(30, 30)

1188: self.rect\_color = constants.RED

1189:

1190:

1191: self.checkammo()

1192: def checkammo(self):

1193: if self.caliber == None:

1194: self.rect = self.rect.inflate(30, 30)

1195: self.rect\_color = constants.RED

1196: projectiles.add(self)

1197: if self.caliber in player.inventory["Ammo"] and player.inventory["Ammo"][self.caliber] > 0:

1198: player.inventory["Ammo"][self.caliber] -= 1

1199: projectiles.add(self)

1200: player.shots += 1

1201: elif self.caliber != None:

1202: self.kill()

1203: projectiles.remove(self)

1204:

1205: def update(self):

1206: if self.caliber is not None:

1207: if self.speed > 0:

1208: for obstacle in map.obstacles:

1209: if self.rect.colliderect(obstacle):

1210: if self.caliber != "50cal":

1211: self.speed = 0

1212: pygame.draw.rect(map.map\_surface, constants.WHITE, self.rect)

1213:

1214: # Calculate time elapsed since the last frame

1215: current\_time = time.time()

1216: time\_elapsed = current\_time - self.alivetime

1217:

1218: # Calculate new position based on time elapsed and speed

1219: distance = self.speed \* time\_elapsed

1220: target\_x, target\_y = self.target\_pos

1221: dx = target\_x - self.rect.x

1222: dy = target\_y - self.rect.y

1223: distance\_to\_target = max(abs(dx), abs(dy))

1224: if distance\_to\_target < self.lastdistance:

1225: self.lastdistance = distance\_to\_target

1226: elif distance\_to\_target > self.lastdistance:

1227: projectiles.remove()

1228: self.kill()

1229: return

1230:

1231: if distance\_to\_target > 0:

1232: speed\_x = (dx / distance\_to\_target) \* distance

1233: speed\_y = (dy / distance\_to\_target) \* distance

1234:

1235:

1236: if distance\_to\_target == 0:

1237: projectiles.remove(self)

1238: self.kill()

1239:

1240: try:

1241: self.rect.x += int(speed\_x)

1242: self.rect.y += int(speed\_y)

1243: except UnboundLocalError:

1244: self.rect.x += 0

1245: self.rect.y += 0

1246:

1247: if self.rect.left < 0 or self.rect.right > map.map\_width or self.rect.top < 0 or self.rect.bottom > map.map\_height:

1248: # Handle if the bullet goes out of bounds

1249: projectiles.remove(self)

1250: self.kill()

1251:

1252: else:

1253: screentext.changetext("OUT OF AMMO")

1254: self.caliber = None

1255: projectiles.remove(self)

1256: self.kill()

1257: else:

1258: # Draw the inflated rect as red

1259: pygame.draw.rect(map.map\_surface, self.rect\_color, self.rect)

1260: if time.time() - self.alivetime > 0.1:

1261: projectiles.remove(self)

1262: self.kill()

1263: class LockpickingGame:

1264: def \_\_init\_\_(self):

1265: self.width, self.height = constants.screen\_width, constants.screen\_height

1266: self.screen = pygame.display.set\_mode((self.width, self.height))

1267: pygame.display.set\_caption("Mini Game")

1268: self.clock = pygame.time.Clock()

1269:

1270: # Load images

1271: self.lock\_image = pygame.image.load(constants.assetpath + "lock.png")

1272: self.lockpick\_image = pygame.image.load(constants.assetpath + "lockpick.png")

1273:

1274: # Set up the lock variables

1275: self.lock\_angle = 0

1276: self.lock\_speed = 0

1277: self.lock\_original\_angle = 0

1278: self.lockpick\_health = 250

1279: self.max\_lockpickhp = 250

1280:

1281: # Set up the lockpick variables

1282: self.lockpick\_angle = 0

1283: self.stop\_lockpick = False

1284:

1285: # Generate a random angle for the lock

1286: self.target\_angle = random.randint(-179, 180)

1287: self.target\_angles = list(range(self.target\_angle - 5, self.target\_angle + 6))

1288: self.target\_angles = [i for i in self.target\_angles if i <= 180]

1289:

1290: #Flags to check win status

1291: self.won = False

1292: self.running = True

1293:

1294: def rotate\_lock(self, direction):

1295: if direction == "right":

1296: self.lock\_speed = 1

1297: self.stop\_lockpick = True

1298: if self.lockpick\_angle // 1 not in self.target\_angles:

1299: self.lockpick\_health -= 1

1300: self.drawhp()

1301:

1302: def stop\_rotating(self):

1303: self.lock\_speed = 0

1304: self.stop\_lockpick = False

1305:

1306: def check\_win(self):

1307: if self.lock\_angle >= 90 and math.isclose(self.lockpick\_angle, self.target\_angle, abs\_tol=5):

1308: return True

1309: return False

1310:

1311: def drawhp(self):

1312: HEALTH\_BAR\_BG\_COLOR = constants.RED

1313: HEALTH\_BAR\_COLOR = constants.GREEN

1314: BORDER\_COLOR = constants.BLACK

1315:

1316: bar\_width = 500

1317: bar\_height = 50

1318: current\_health\_width = int((self.lockpick\_health / self.max\_lockpickhp) \* bar\_width)

1319: health\_bar\_x = (constants.screen\_width//2) - 250

1320: health\_bar\_y = 52 - bar\_height - 2

1321:

1322: pygame.draw.rect(self.screen, HEALTH\_BAR\_BG\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height))

1323: pygame.draw.rect(self.screen, HEALTH\_BAR\_COLOR, (health\_bar\_x, health\_bar\_y, current\_health\_width, bar\_height))

1324: pygame.draw.rect(self.screen, BORDER\_COLOR, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height), 1)

1325:

1326: def run(self,ID):

1327: global lockedcontainers

1328: while self.running:

1329: self.screen.fill(constants.DARK\_GRAY)

1330: for event in pygame.event.get():

1331: if event.type == pygame.QUIT:

1332: self.running = False

1333:

1334: keys = pygame.key.get\_pressed()

1335: if keys[pygame.K\_d]:

1336: self.rotate\_lock("right")

1337: else:

1338: self.stop\_rotating()

1339:

1340: if self.lockpick\_health <= 0:

1341: if "Lockpick" in player.inventory.get("Misc"):

1342: player.inventory["Misc"]["Lockpick"] -= 1

1343: RMV["AMMO"] = ("Lockpick")

1344: RMV["AMOUNT"] = 1

1345: s.send((json.dumps(RMV) + "#").encode())

1346: if player.inventory["Misc"]["Lockpick"] <= 0:

1347: del player.inventory["Misc"]["Lockpick"]

1348: self.lockpick\_health = self.max\_lockpickhp

1349: else:

1350: self.running = False

1351:

1352: self.lock\_angle += self.lock\_speed

1353: if self.lock\_angle >= 90:

1354: self.lock\_angle = 90

1355:

1356: self.drawhp()

1357:

1358: if not self.stop\_lockpick:

1359: self.lockpick\_angle = math.degrees(

1360: math.atan2(pygame.mouse.get\_pos()[1] - self.height / 2, pygame.mouse.get\_pos()[0] - self.width / 2)

1361: )

1362:

1363: if self.check\_win():

1364: if self.lock\_angle >= 90:

1365: if not self.won:

1366: self.won = True

1367: else:

1368: pygame.draw.rect(self.screen, constants.RED, (self.width // 2 - 50, self.height // 2 - 50, 100, 100))

1369:

1370: if not self.won:

1371: rotated\_lock = pygame.transform.rotate(self.lock\_image, -1 \* self.lock\_angle)

1372: else:

1373: rotated\_lock = pygame.transform.rotate(self.lock\_image, 270)

1374: ULK["ID"] = ID

1375: s.send((json.dumps(ULK) + "#").encode())

1376:

1377: lockedcontainers.remove(ID)

1378:

1379: openCont(ID)

1380: self.running = False

1381:

1382: rotated\_lockpick = pygame.transform.rotate(self.lockpick\_image, -1 \* self.lockpick\_angle)

1383:

1384: lock\_rect = rotated\_lock.get\_rect(center=(self.width // 2, self.height // 2))

1385: lockpick\_rect = rotated\_lockpick.get\_rect(center=(self.width // 2, self.height // 2))

1386:

1387: self.screen.blit(rotated\_lock, lock\_rect)

1388: self.screen.blit(rotated\_lockpick, lockpick\_rect)

1389:

1390: if not keys[pygame.K\_d] and self.lock\_angle > self.lock\_original\_angle:

1391: self.lock\_angle -= 0.5

1392:

1393: if self.lock\_speed != 0:

1394: self.lockpick\_moving = True

1395: else:

1396: self.lockpick\_moving = False

1397:

1398: pygame.display.flip()

1399: self.clock.tick(60)

1400: class InventoryMenu:

1401: def \_\_init\_\_(self, screen\_width, screen\_height, player):

1402: pygame.init()

1403: self.screen = pygame.display.set\_mode((screen\_width, screen\_height))

1404: self.clock = pygame.time.Clock()

1405: self.font = pygame.font.Font(None, 28)

1406: self.player\_inventory = player.inventory

1407: self.player\_equipped = player.equipped

1408: self.menu\_width = screen\_width

1409: self.menu\_height = screen\_height

1410: self.menu\_x = (screen\_width - self.menu\_width) // 2

1411: self.menu\_y = (screen\_height - self.menu\_height) // 2

1412: self.selected\_tab = 'Weapons'

1413: self.selected\_items = {} # Dictionary to store selected items for each tab

1414: self.item\_rects = {} # Dictionary to store item rects

1415: self.is\_running = True

1416: self.use\_button\_visible = False

1417:

1418: # Define the positions and sizes of the black squares

1419: self.equipped\_x = self.menu\_x + self.menu\_width - 50

1420: self.equipped\_y = (self.menu\_y + self.menu\_height) // 2 - (40) \* 2 # Adjusted position

1421: self.equipped\_size = 30

1422: self.equipped\_spacing = 10

1423: self.checkforzero()

1424:

1425: def checkforzero(self):

1426: try:

1427: for i in player.inventory["Ammo"]:

1428: if player.inventory["Ammo"][i] <= 0:

1429: del player.inventory["Ammo"][i]

1430: except RuntimeError:

1431: pass

1432:

1433: def show\_inventory\_menu(self):

1434: use\_button\_rect = pygame.Rect(

1435: (self.menu\_x + self.menu\_width) // 2 - 50,

1436: (self.menu\_y + self.menu\_height) // 2 - 25,

1437: 100,

1438: 50

1439: )

1440:

1441: while self.is\_running:

1442: clicked\_tab = None

1443: clicked\_item = None

1444: use\_button\_clicked = False # Flag to track if the "Use" button was clicked

1445:

1446: for event in pygame.event.get():

1447: if event.type == pygame.KEYDOWN:

1448: if event.key == pygame.K\_ESCAPE:

1449: self.is\_running = False

1450: elif event.type == pygame.MOUSEBUTTONDOWN:

1451: if event.button == 1: # Left mouse button

1452: clicked\_tab = self.get\_clicked\_tab(event.pos)

1453: clicked\_item = self.get\_clicked\_item(event.pos)

1454: if use\_button\_rect.collidepoint(event.pos):

1455: use\_button\_clicked = True # Set the flag if the "Use" button is clicked

1456:

1457: if clicked\_tab is not None:

1458: self.selected\_tab = clicked\_tab

1459: self.selected\_items[self.selected\_tab] = None # Reset selected item when changing tabs

1460: self.use\_button\_visible = False # Hide the "Use" button when switching tabs

1461:

1462: if clicked\_item is not None:

1463: self.selected\_items[self.selected\_tab] = clicked\_item

1464: # If the selected item is in Weapons, Armor, or Consumables tab, show the Use button

1465: if self.selected\_tab in ["Weapons", "Armor", "Consumables"]:

1466: self.use\_button\_visible = True

1467: else:

1468: self.use\_button\_visible = False

1469:

1470: # Move the item rects initialization outside the tab selection check

1471: self.initialize\_item\_rects()

1472:

1473: self.screen.fill(constants.DARK\_GRAY)

1474: pygame.draw.rect(self.screen, (constants.DARK\_GRAY),

1475: (self.menu\_x, self.menu\_y, self.menu\_width, self.menu\_height))

1476:

1477: self.draw\_tabs()

1478: self.draw\_inventory()

1479: self.draw\_selected\_item()

1480:

1481: if self.use\_button\_visible:

1482: self.show\_use\_button()

1483: # Check if the "Use" button is clicked

1484: if use\_button\_clicked:

1485: self.use\_item(self.selected\_items[self.selected\_tab])

1486:

1487: # Call the draw\_equipped function to draw the black squares

1488: self.draw\_equipped()

1489:

1490: for enemy in enemysprites.sprites():

1491: enemy.check\_collision(player)

1492:

1493: pygame.display.flip()

1494: self.clock.tick(60)

1495:

1496: def initialize\_item\_rects(self):

1497: self.item\_rects = {}

1498: text\_x = self.menu\_x + 20

1499: text\_y = self.menu\_y + 50

1500: selected\_items = self.player\_inventory.get(self.selected\_tab, {})

1501:

1502: for item, quantity in selected\_items.items():

1503: item\_rect = pygame.Rect(text\_x, text\_y, self.menu\_width - 40, 30)

1504: self.item\_rects[item] = item\_rect # Store item rect in the dictionary

1505: text\_y += 50 # Increase vertical spacing between items

1506:

1507: def draw\_tabs(self):

1508: tab\_width = self.menu\_width // len(self.player\_inventory)

1509: tab\_height = 30

1510: text\_y = self.menu\_y + 10

1511: for i, tab in enumerate(self.player\_inventory.keys()):

1512: tab\_rect = pygame.Rect(self.menu\_x + (tab\_width \* i), text\_y, tab\_width, tab\_height)

1513: tab\_text = self.font.render(tab, True, (0, 0, 0))

1514: pygame.draw.rect(self.screen, (200, 200, 200), tab\_rect)

1515: self.screen.blit(tab\_text, (tab\_rect.x + 10, tab\_rect.y + 5))

1516: if self.selected\_tab == tab:

1517: pygame.draw.rect(self.screen, (0, 0, 255), tab\_rect, 3)

1518:

1519: def draw\_inventory(self):

1520: text\_x = self.menu\_x + 20

1521: text\_y = self.menu\_y + 50

1522: selected\_items = self.player\_inventory.get(self.selected\_tab, {})

1523:

1524: for item, quantity in selected\_items.items():

1525: item\_rect = self.item\_rects.get(item, pygame.Rect(text\_x, text\_y, self.menu\_width - 40, 30))

1526: item\_text = self.font.render(f'{item}: {quantity}', True, (0, 0, 0))

1527: pygame.draw.rect(self.screen, (200, 200, 200), item\_rect)

1528: self.screen.blit(item\_text, (text\_x + 10, text\_y + 5))

1529: text\_y += 50 # Increase vertical spacing between items

1530:

1531: def draw\_selected\_item(self):

1532: selected\_item = self.selected\_items.get(self.selected\_tab, None)

1533: if selected\_item:

1534: text\_x = self.menu\_x + 20

1535: text\_y = self.menu\_y + 50

1536:

1537: try:

1538: if hasattr(items, self.selected\_tab):

1539: item\_data = None

1540: if self.selected\_tab == "Weapons":

1541: item\_data = items.Weapons.get(selected\_item)

1542: elif self.selected\_tab == "Armor":

1543: item\_data = items.Armor.get(selected\_item)

1544: elif self.selected\_tab == "Ammo":

1545: item\_data = items.Ammo.get(selected\_item)

1546: elif self.selected\_tab == "Consumables":

1547: item\_data = items.Consumables.get(selected\_item)

1548: elif self.selected\_tab == "Misc":

1549: item\_data = items.Misc.get(selected\_item)

1550:

1551: if item\_data:

1552: item = selected\_item

1553: img\_path = (item\_data[0])

1554: img = pygame.image.load(constants.assetpath+img\_path)

1555: imgrect = img.get\_rect()

1556: imgrect.center = (self.menu\_x + self.menu\_width // 2, self.menu\_y + self.menu\_height - 30)

1557: self.screen.blit(img, imgrect)

1558:

1559: text\_y += self.menu\_height // 2

1560:

1561: item\_details\_text = self.font.render(f'Name: {item}', True, (0, 0, 0))

1562: self.screen.blit(item\_details\_text, (text\_x, text\_y))

1563: text\_y += 30

1564:

1565: if self.selected\_tab == "Weapons":

1566: dmg = item\_data[1]

1567: firerate = item\_data[2]

1568: ammo = item\_data[3]

1569: details = [f'Damage: {dmg}', f'Firerate: {firerate}', f'Ammo Type: {ammo}']

1570: elif self.selected\_tab == "Armor":

1571: slot = item\_data[1]

1572: armorval = item\_data[2]

1573: details = [f'Slot: {slot}', f'Armor Value: {armorval}']

1574: elif self.selected\_tab == "Ammo":

1575: bulletvel = item\_data[1]

1576: penval = item\_data[2]

1577: details = [f'Bullet Velocity: {bulletvel}', f'Penetration Value: {penval}']

1578: elif self.selected\_tab == "Consumables":

1579: effect = item\_data[1]

1580: value = item\_data[2]

1581: details = [f'Effect: {effect}', f'Effect Amount: {value}']

1582: elif self.selected\_tab == "Misc":

1583: MiscID = item\_data[1]

1584: details = [f'ID: {MiscID}']

1585:

1586: for detail in details:

1587: detail\_text = self.font.render(detail, True, (0, 0, 0))

1588: self.screen.blit(detail\_text, (text\_x, text\_y))

1589: text\_y += 30

1590:

1591: selected\_item\_text = self.font.render(f'Selected Item: {selected\_item}', True, (0, 0, 0))

1592: text\_y = self.menu\_y + self.menu\_height - 30

1593: self.screen.blit(selected\_item\_text, (text\_x, text\_y))

1594:

1595: else:

1596: print(f"'{self.selected\_tab}' not found in items.")

1597: except Exception as e:

1598: print(e)

1599: pass

1600:

1601: def show\_use\_button(self):

1602: # Calculate the position for the centered "Use" button

1603: button\_width = 100

1604: button\_height = 50

1605: button\_x = (self.menu\_x + self.menu\_width) // 2 - (button\_width // 2)

1606: button\_y = (self.menu\_y + self.menu\_height) // 2 - (button\_height // 2)

1607:

1608: # Create a green "Use" button

1609: use\_button\_rect = pygame.Rect(button\_x, button\_y, button\_width, button\_height)

1610: pygame.draw.rect(self.screen, (0, 255, 0), use\_button\_rect)

1611: use\_text = self.font.render("USE", True, (0, 0, 0))

1612: use\_text\_rect = use\_text.get\_rect(center=use\_button\_rect.center)

1613: self.screen.blit(use\_text, use\_text\_rect)

1614:

1615: def draw\_equipped(self):

1616: equipped\_slots = player.equipped.keys()

1617: for i, slot in enumerate(equipped\_slots):

1618: item\_name = player.equipped[slot]

1619: if item\_name:

1620: if item\_name in player.equipped\_images:

1621: img = player.equipped\_images[item\_name]

1622: else:

1623: # Load the image for the equipped item

1624: if self.selected\_tab == "Weapons":

1625: item\_data = items.Weapons.get(item\_name)

1626: elif self.selected\_tab == "Armor":

1627: item\_data = items.Armor.get(item\_name)

1628: elif self.selected\_tab == "Consumables":

1629: item\_data = items.Consumables.get(item\_name)

1630:

1631: try:

1632: img\_path = (item\_data[0])

1633: img = pygame.image.load(constants.assetpath+img\_path)

1634: img = pygame.transform.scale(img, (self.equipped\_size, self.equipped\_size))

1635: player.equipped\_images[item\_name] = img # Store the image in the dictionary

1636:

1637: except Exception as e:

1638: print(e)

1639: pass

1640:

1641: # Scale the image to the size of the equipped square

1642: try:

1643: img = pygame.transform.scale(img, (self.equipped\_size, self.equipped\_size))

1644:

1645: # Display the equipped item's image

1646: rect\_x = self.equipped\_x

1647: rect\_y = self.equipped\_y + (i \* (self.equipped\_size + self.equipped\_spacing))

1648: rect = pygame.Rect(rect\_x, rect\_y, self.equipped\_size, self.equipped\_size)

1649: pygame.draw.rect(self.screen, (0, 0, 0), rect)

1650: imgrect = img.get\_rect(topleft=(rect\_x, rect\_y))

1651: self.screen.blit(img, imgrect)

1652: except Exception as e:

1653: print(e)

1654: else:

1655: # If no item is equipped in this slot, draw an empty black square

1656: rect\_x = self.equipped\_x

1657: rect\_y = self.equipped\_y + (i \* (self.equipped\_size + self.equipped\_spacing))

1658: rect = pygame.Rect(rect\_x, rect\_y, self.equipped\_size, self.equipped\_size)

1659: pygame.draw.rect(self.screen, (0, 0, 0), rect)

1660:

1661: def get\_clicked\_tab(self, mouse\_pos):

1662: tab\_width = self.menu\_width // len(self.player\_inventory)

1663: tab\_height = 30

1664: text\_y = self.menu\_y + 10

1665: for i, tab in enumerate(self.player\_inventory.keys()):

1666: tab\_rect = pygame.Rect(self.menu\_x + (tab\_width \* i), text\_y, tab\_width, tab\_height)

1667: if tab\_rect.collidepoint(mouse\_pos):

1668: return tab

1669:

1670: return None

1671:

1672: def get\_clicked\_item(self, mouse\_pos):

1673: for item, item\_rect in self.item\_rects.items():

1674: if item\_rect.collidepoint(mouse\_pos):

1675: return item

1676:

1677: def save\_equipped\_items(self,player):

1678: equipped\_data = {

1679: "H": player.equipped["H"],

1680: "C": player.equipped["C"],

1681: "F": player.equipped["F"],

1682: "Hand": player.equipped["Hand"],

1683: "Offhand": player.equipped["Offhand"]

1684: }

1685:

1686: with open("equipped\_items.json", "w") as json\_file:

1687: json.dump(equipped\_data, json\_file)

1688: def use\_item(self, item):

1689: if self.selected\_tab == "Weapons":

1690: item\_data = items.Weapons.get(item)

1691: player.equipped["Hand"] = item

1692: player.equipped\_images[item] = pygame.image.load(constants.assetpath+item\_data[0])

1693: if self.selected\_tab == "Armor":

1694: item\_data = items.Armor.get(item)

1695: player.equipped[item\_data[1]] = item

1696: player.equipped\_images[item] = pygame.image.load(constants.assetpath+item\_data[0])

1697: player.statupdate()

1698: if self.selected\_tab == "Consumables":

1699: item\_data = items.Consumables.get(item)

1700: player.equipped["Offhand"] = item

1701: player.equipped\_images[item] = pygame.image.load(constants.assetpath+item\_data[0])

1702: self.save\_equipped\_items(player)

1703: print("equipped")

1704: def recv\_server():

1705: '''

1706: This is called via a thread in which the client activley and indepentaley recieves data from the server

1707: irrelevant to the current going ons, this is mainly used to receive quest information as well as initilazing your inventory

1708: and players overall data.

1709:

1710: The use of a # is present at the end of a packet to act as a header/footer to ensure the client only recieves the information in that packet

1711: and also allows ordering of the packets meanign if multipel packets were recieved simultaneously only they would be executed in order

1712: :return:

1713: '''

1714: global pmap #sets global vairbales for the main loop to access

1715: global xpos

1716: global ypos

1717: global hp

1718: global map

1719: global enemies

1720: global interrupt

1721:

1722: while True:

1723: data = s.recv(2048)

1724: try:

1725: data\_list = data.decode().split("#") #use as a packet header and footer

1726: for msg\_str in data\_list:

1727: if msg\_str != "":

1728: msg = json.loads(msg\_str) #decode the msg

1729: #All the commands related to the command header of the packet

1730: if msg["COMMAND"] == "INIT":

1731: xpos = msg["X"]

1732: ypos = msg["Y"]

1733: hp = msg["HP"]

1734: pmap = msg["map"]

1735: print("PLAYERINTIIALZED")

1736: if msg["COMMAND"] == "QUEST":

1737: for q, v in player.quests:

1738: if q == msg["Q"]:

1739: # Update the existing value

1740: player.quests.remove((q, v))

1741: player.quests.add((q, msg["V"]))

1742: break

1743: else:

1744: # Add a new (QUEST, VALUE) pair if it doesn't exist

1745: player.quests.add((msg["Q"],msg["V"]))

1746: if msg["COMMAND"] == "DRAW":

1747: sid = int(msg["ID"])

1748: RECTX = int(msg["RECTX"])

1749: RECTY = int(msg["RECTY"])

1750: drawotherplayers(sid,RECTX,RECTY)

1751: if msg["COMMAND"] == "KILL":

1752: sid = msg["ID"]

1753: for sprite in otherplayers:

1754: if sprite.sid == sid:

1755: otherplayers.remove(sid)

1756: sprite.kill()

1757: sprite\_ids.remove(sid)

1758: if msg["COMMAND"] == "EU":

1759: identifier = msg["ID"]

1760: hp = msg["HP"]

1761: rectx = msg["RECTX"]

1762: recty = msg["RECTY"]

1763: name = msg["NAME"]

1764: enemymap = msg["MAP"]

1765: Enemy.from\_csv(name, rectx, recty, identifier,hp,enemymap)

1766: if msg["COMMAND"] == "WE":

1767: player.continv["Weapons"] = json.loads(msg["I"])

1768: if msg["COMMAND"] == "AR":

1769: player.continv["Armor"] = json.loads(msg["I"])

1770: if msg["COMMAND"] == "AM":

1771: player.continv["Ammo"] = json.loads(msg["I"])

1772: if msg["COMMAND"] == "CS":

1773: player.continv["Consumables"] = json.loads(msg["I"])

1774: if msg["COMMAND"] == "MI":

1775: player.continv["Misc"] = json.loads(msg["I"])

1776: if msg["COMMAND"] == "PWE":

1777: player.inventory["Weapons"] = json.loads(msg["I"])

1778: if msg["COMMAND"] == "PAR":

1779: player.inventory["Armor"] = json.loads(msg["I"])

1780: if msg["COMMAND"] == "PAM":

1781: player.inventory["Ammo"] = json.loads(msg["I"])

1782: if msg["COMMAND"] == "PCS":

1783: player.inventory["Consumables"] = json.loads(msg["I"])

1784: if msg["COMMAND"] == "PMI":

1785: player.inventory["Misc"] = json.loads(msg["I"])

1786: load\_equipped(player)

1787: if msg["COMMAND"] == "REWARD":

1788: item = msg["ITEM"]

1789: amount = msg["A"]

1790: sender = msg["SNDR"]

1791: screentext.quest(item,amount,sender)

1792: if msg["COMMAND"] == "LCK":

1793: lockedcontainers.append(int(msg["ID"]))

1794: if msg["COMMAND"] == "SULK":

1795: try:

1796: lockedcontainers.remove(int(msg["ID"]))

1797: except IndexError:

1798: pass #cotnainer has alrdy been delcared as unlocked

1799: if msg["COMMAND"] == "PNG":

1800: s.send((json.dumps(PING) + "#").encode())

1801:

1802:

1803: except Exception as e:

1804: print(e) #prints any errors and attempts to continue

1805: pass

1806:

1807: threading.Thread(target=recv\_server, args=()).start() #begin the autosave thread

1808: def drawotherplayers(sid,RECTX,RECTY):

1809: '''

1810: simply a function to take info recieved from a packet and add the otherplayer to a class instance an store the id in a list

1811: :param sid:

1812: :param RECTX:

1813: :param RECTY:

1814: :param MAPID:

1815: :return:

1816: '''

1817: for sprite in otherplayers:

1818: if sid in sprite\_ids:

1819: sprite.rect.x = RECTX

1820: sprite.rect.y = RECTY

1821: else:

1822: sprite\_ids.append(sid)

1823: sid = OTHERPLAYER(sid, RECTX, RECTY, "walk 1.png")

1824: otherplayers.add(sid)

1825: def get\_object\_rects(filename):

1826: '''

1827: Gathers all objects from a specified tmx file and returns them for the map class.

1828: :param filename:

1829: :return: object\_rects

1830: '''

1831: tmx\_data = pytmx.util\_pygame.load\_pygame(filename)

1832: object\_rects = []

1833: for layer in tmx\_data.visible\_layers:

1834: if isinstance(layer, pytmx.TiledObjectGroup):

1835: for obj in layer:

1836: object\_rects.append(pygame.Rect(obj.x, obj.y, obj.width, obj.height))

1837: return object\_rects

1838: def get\_interactable\_rects(filename):

1839: '''

1840: similar to onject rects but only returns interactable ones

1841: :param filename:

1842: :return: interactable\_rects

1843: '''

1844: tmx\_data = pytmx.util\_pygame.load\_pygame(filename)

1845: interactable\_rects = []

1846: for layer in tmx\_data.visible\_layers:

1847: if isinstance(layer, pytmx.TiledObjectGroup):

1848: for obj in layer:

1849: if obj.properties.get('interactable', False):

1850: interType = obj.properties.get('intertype')

1851: interID = obj.properties.get('interactable')

1852: interRect = pygame.Rect(obj.x, obj.y, obj.width, obj.height)

1853: interactable\_rects.append((interType,interRect,interID))

1854: return interactable\_rects

1855: def draw\_health\_bar(screen, player):

1856: """

1857: draws the healthbar on a specified layer using the player as a paramater

1858: :param screen:

1859: :param player:

1860: :return:

1861: """

1862:

1863: # Define sizes and positions

1864: bar\_width = 100

1865: bar\_height = 20

1866: bar\_padding = 2

1867: health\_ratio = player.hp / player.max\_hp

1868:

1869: # Calculate position of health bar

1870: health\_bar\_x = screen.get\_width() - bar\_width - bar\_padding

1871: health\_bar\_y = bar\_padding

1872:

1873: # Load the image

1874: image\_path = constants.assetpath + "dmgreduc.png"

1875: image = pygame.image.load(image\_path)

1876:

1877: # Scale the image up by 2x

1878: image = pygame.transform.scale(image, (image.get\_width() \* 2, image.get\_height() \* 2))

1879:

1880: # Calculate image position

1881: image\_x = health\_bar\_x + (bar\_width - image.get\_width()) / 2

1882: image\_y = health\_bar\_y + bar\_height + bar\_padding

1883:

1884: # Draw the scaled image

1885: screen.blit(image, (image\_x, image\_y))

1886:

1887: # Draw background of the health bar

1888: pygame.draw.rect(screen, constants.WHITE, (health\_bar\_x, health\_bar\_y, bar\_width, bar\_height))

1889:

1890: # Draw current health

1891: health\_bar\_width = health\_ratio \* bar\_width

1892: health\_bar\_rect = pygame.Rect(health\_bar\_x, health\_bar\_y, health\_bar\_width, bar\_height)

1893: pygame.draw.rect(screen, constants.GREEN, health\_bar\_rect)

1894:

1895: # Create a text surface

1896: text\_surface = font.render(str(player.damage\_reduc), True, constants.WHITE)

1897: hptext\_surface = font.render(("HP:"+str(player.hp)), True, constants.BLUE)

1898:

1899: # Calculate text position

1900: text\_x = image\_x + 30

1901: text\_y = image\_y \*2.8

1902:

1903: # Draw the text surface

1904: screen.blit(text\_surface, (text\_x, text\_y))

1905: screen.blit(hptext\_surface, (health\_bar\_rect))

1906: def draw\_equipped\_items(screen, player):

1907: '''

1908: Draws the equipped items in 4 black squares utilizing the player instances class variables.

1909: :param screen:

1910: :param player:

1911: :return:

1912: '''

1913: equipped\_slots = player.equipped.keys()

1914:

1915: for i, slot in enumerate(equipped\_slots): #Enumerates the slots uses contant values to seperate each square slightly

1916: item\_name = player.equipped[slot]

1917:

1918: # Draw black square

1919: rect\_x = constants.equipped\_x

1920: rect\_y = constants.equipped\_y + (i \* (constants.equipped\_size + constants.equipped\_spacing))

1921: rect = pygame.Rect(rect\_x, rect\_y, constants.equipped\_size, constants.equipped\_size)

1922: pygame.draw.rect(screen, constants.WHITE, rect)

1923:

1924: if item\_name != {}: # Ensures the item isn't blank.

1925: if item\_name in player.equipped\_images:

1926: img = player.equipped\_images[item\_name]

1927: img = pygame.transform.scale(img, (constants.equipped\_size, constants.equipped\_size))

1928: imgrect = img.get\_rect(topleft=(rect\_x, rect\_y)) #uses topleft function to get the position for the squares.

1929: screen.blit(img, imgrect) #blits the image

1930: def spawn\_NPCS(map):

1931: '''

1932: Returns nothing, simply creates the NPC instances upon changing maps

1933: :param map:

1934: :return None:

1935: '''

1936: with open(constants.configpath+'spawn.csv', 'r') as file:

1937: reader = csv.DictReader(file)

1938: # Loop through each row in the file

1939: for row in reader:

1940: # Check if the name in the row matches the name of the enemy we're looking for

1941: if row['mapID'] == map:

1942: mapid = str(row["mapID"])

1943: name = str(row["name"])

1944: rectx = int(row['rectx'])

1945: recty = int(row['recty'])

1946: image = (constants.assetpath+ str(row["image"]))

1947:

1948: npc = NPC(image,mapid,rectx,recty,name)

1949:

1950: # Return None if we couldn't find an enemy with the given name

1951: return None

1952: def cleanup():

1953: '''

1954: Cleans up enemys that are either dead or exist on a different layer

1955: Removes other players from being diplayed that aren't on the layer you're on.

1956: :return:

1957: '''

1958: for enemy in enemysprites:

1959: if enemy.health <= 0:

1960: enemies.remove(enemy.ID)

1961: enemy.alive = False

1962: enemysprites.remove(enemy)

1963: def openCont(ID):

1964: '''

1965: Opens a container using the specified ID, it checks it isnt locked and then creates the transfer class

1966: :param ID:

1967: :return:

1968: '''

1969: global lockedcontainers

1970: """

1971: :param ID:

1972: :return:

1973: """

1974: ID = int(ID)

1975: if ID in lockedcontainers:

1976: if "Lockpick" in player.inventory.get("Misc"):

1977: player.inventory["Misc"]["Lockpick"] -= 1

1978: RMV["AMMO"] = ("Lockpick")

1979: RMV["AMOUNT"] = 1

1980: s.send((json.dumps(RMV) + "#").encode())

1981: if player.inventory["Misc"]["Lockpick"] <= 0:

1982: del player.inventory["Misc"]["Lockpick"]

1983: l = LockpickingGame()

1984: l.run(ID)

1985: else:

1986: screentext.changetext("LOCKED")

1987: else:

1988: CONT["ID"] = ID

1989: s.send((json.dumps(CONT) + "#").encode())

1990: player.contID = ID

1991: transfer = Transfer(player)

1992: transfer.run()

1993: def changemaps(id):

1994: '''

1995: Doesnt return anything in particular but essentially redefines the map class with the new maps vlaues and instances

1996: :param id:

1997: :return:

1998: '''

1999: """

2000: :param id:

2001: :return:

2002: """

2003: player.map = id

2004: tmx = map\_dict[id][0]

2005: csv = map\_dict[id][1]

2006: global map

2007: map = Map(tmx, csv)

2008: map.changemap(id)

2009: spawn\_NPCS(id)

2010: time.sleep(0.7)

2011: for i in otherplayers:

2012: if i.sid != -1:

2013: i.kill()

2014: sprite\_ids.remove(i.sid)

2015:

2016: print("CHANGED MAPS TO MAP ID",id,"\n","NPCS ON THIS LEVEL:",len(NPCS),"\n","INTERACTABLES ON THIS LEVEL:",len(map.interactables),"total enemies",len(enemysprites))

2017: def update\_camera(player):

2018: '''

2019: Render function essentially applys camera offsets fot the x.y values

2020: :param player:

2021: :return X and Y offsets for camera rendering:

2022: '''

2023:

2024: # adjust camera offset to keep player centered

2025: x = (screen\_width // 2 - player.rect.centerx)

2026: y = (screen\_height // 2 - player.rect.centery)

2027:

2028: # apply camera limits

2029: x = min(0, x)

2030: x = max(-(map.map\_width - screen\_width), x)

2031: y = min(0, y)

2032: y = max(-(map.map\_height - screen\_height), y)

2033:

2034: return x, y

2035: def render(screen, map\_surface, camera\_x, camera\_y,player):

2036: """

2037: Renders the screen using the map surface and camera offsets as well as the player instance

2038: Also handles drawing the onscreen set as well as drawing the equipped items.

2039: :param screen:

2040: :param map\_surface:

2041: :param camera\_x:

2042: :param camera\_y:

2043: :param player:

2044: :return:

2045: """

2046: # Adjust player position relative to camera

2047: player.rect.x += camera\_x

2048: player.rect.y += camera\_y

2049:

2050: # Draw map surface and player

2051: screen.blit(map\_surface, (camera\_x, camera\_y))

2052: screen.blit(player.image, player.rect)

2053:

2054:

2055: player.rect.x -= camera\_x

2056: player.rect.y -= camera\_y

2057: screentext.draw(time.time())

2058: screentext.changetext("")

2059:

2060: draw\_equipped\_items(screen,player)

2061: def load\_equipped(player):

2062: '''

2063: Loads the equipped items from the stored JSON file if it exists

2064: :param player:

2065: :return:

2066: '''

2067: try:

2068: with open("equipped\_items.json", "r") as json\_file:

2069: equipped\_data = json.load(json\_file)

2070:

2071: for slot, item in equipped\_data.items():

2072: if slot in ["H", "C", "F"]:

2073: if item != {}:

2074: armor\_data = items.Armor.get(item)

2075: if armor\_data:

2076: if item in player.inventory["Armor"]:

2077: player.equipped[slot] = item

2078: player.equipped\_images[item] = pygame.image.load(constants.assetpath+armor\_data[0])

2079: player.statupdate()

2080: elif slot == "Hand":

2081: if item != {}:

2082: weapon\_data = items.Weapons.get(item)

2083: if weapon\_data:

2084: if item in player.inventory["Weapons"]:

2085: player.equipped["Hand"] = item

2086: player.equipped\_images[item] = pygame.image.load(constants.assetpath+weapon\_data[0])

2087: elif slot == "Offhand":

2088: if item != {}:

2089: consumable\_data = items.Consumables.get(item)

2090: if consumable\_data:

2091: if item in player.inventory["Consumables"]:

2092: player.equipped["Offhand"] = item

2093: player.equipped\_images[item] = pygame.image.load(constants.assetpath+consumable\_data[0])

2094:

2095: except FileNotFoundError:

2096: with open("equipped\_items.json", "w") as equippeditemfile:

2097: equippeditemfile.write("""""{"H":"","C":"","F": "","Hand": "", "Offhand": ""}""""")

2098: print("Equipped items file created")

2099: equippeditemfile.close()

2100: pass

2101:

2102: except Exception as e:

2103: print(e)

2104: pass

2105: def save():

2106: '''

2107: Self explanatory this is the infinite loop called by the autosave thread that simply pakcages and sends the server data on a 0.5s delay

2108: :return Returns nothing but coudl argue it returns a DATA pakcet to be sent to the server:

2109: '''

2110: while True:

2111: DATA["COMMAND"] = "GENERAL"

2112: DATA["X"] = (str(player.rect.x // 32))

2113: DATA["Y"] = (str(player.rect.y // 64))

2114: DATA["HP"] = str(player.hp)

2115: DATA["RECTX"] = str((player.rect.x))

2116: DATA["RECTY"] = str((player.rect.y))

2117: DATA["Map"] = player.map

2118: s.send((json.dumps(DATA) + "#").encode())

2119: time.sleep(0.5)

2120:

2121: map = Map('PYGAME.tmx','test\_PATHING.csv')

2122: map\_id = 0

2123: time\_factor = 1

2124:

2125: def playerinit():

2126: '''

2127: Inital function called to create the player instance once recv server has recieved the correct info required

2128: Also changes maps to the correct one and calls the intro menu

2129: :return:

2130: '''

2131: global pmap

2132: global player

2133: try:

2134: time.sleep(0.3)

2135: player = Player(xpos \* 32, ypos \* 64, pmap)

2136: player.hp = hp

2137: except Exception as e:

2138: menu = Menu(screen)

2139: menu.run()

2140: # intro()

2141: player = Player(0 \* 32, 0 \* 64, 0)

2142: menu = Menu(screen)

2143: menu.run()

2144: s.send((json.dumps(INIT) + "#").encode())

2145: threading.Thread(target=save).start()

2146: changemaps(str(player.map))

2147: menu = Menu(screen)

2148: menu.run()

2149: main(player)

2150:

2151: def main(player):

2152: '''

2153: Main function loop

2154: Handles all the main function logic and handles all of the game aspect

2155: :param player:

2156: :return Doesnt return anything its a loop:

2157: '''

2158: global fpressed

2159: global camera\_x

2160: global camera\_y

2161: pygame.init()

2162: """

2163: This is the main loop of the game

2164: :param player:

2165: :return:

2166: """

2167: running = True

2168: try:

2169: mixer.music.load('music.mp3')

2170: mixer.music.play()

2171: except:

2172: pass

2173: while running:

2174: cleanup()

2175: clock.tick(60 \* time\_factor)

2176: keys = pygame.key.get\_pressed()

2177: player.update(keys)

2178: projectiles.update()

2179: enemysprites.update()

2180: otherplayers.draw(map.map\_surface)

2181: for enemy in enemysprites.sprites():

2182: enemy.check\_collision(player)

2183: for enemy in enemysprites:

2184: enemy.draw\_health\_bar(map.map\_surface)

2185: if enemy.map != map.map\_id:

2186: enemies.remove(enemy.ID)

2187: enemy.kill()

2188: else:

2189: map.map\_surface.blit(enemy.image, enemy.rect)

2190: for npc in NPCS:

2191: if npc.map\_id == map.map\_id:

2192: NPCS.draw(map.map\_surface)

2193: npc.check\_collision(player,keys)

2194: camera\_x, camera\_y = update\_camera(player)

2195: screen.fill((255, 255, 255))

2196: #Render The Screen At The End OF Each Run LOOP

2197: render(screen, map.map\_surface, camera\_x, camera\_y,player)

2198: draw\_health\_bar(screen, player)

2199:

2200:

2201: #Picks up any events

2202: for event in pygame.event.get():

2203: if event.type == pygame.QUIT:

2204: DATA["COMMAND"] = "GENERAL"

2205: DATA["X"] = (str(player.rect.x // 32))

2206: DATA["Y"] = (str(player.rect.y // 64))

2207: DATA["HP"] = player.hp

2208: DATA["RECTX"] = (player.rect.x)

2209: DATA["RECTY"] = (player.rect.y)

2210: s.send((json.dumps(DATA) + "#").encode())

2211: print("SAFELEY CLOSED")

2212: s.close()

2213: pygame.quit()

2214: sys.exit()

2215: if event.type == pygame.KEYUP and event.key == pygame.K\_f:

2216: fpressed = False

2217:

2218:

2219:

2220: #Draws The Map Without the Pathing Layer

2221: for layer in map.tmx\_data.visible\_layers:

2222: if layer.name not in ["PATHING","Objects"]: # Skip the PATHING and object layer

2223: for x, y, image in layer.tiles():

2224: map.map\_surface.blit(image, (x \* map.tmx\_data.tilewidth, y \* map.tmx\_data.tileheight))

2225:

2226: pygame.display.update()

2227: pygame.quit()

2228:

2229: time.sleep(2)

2230: playerinit()

2231:

2232:

2233: #SPRITE AND CHARACTER CREDITS TO Bethesda Softworks/Bethesda Game Studios

2234: #INTRO VIDO SEQUENCE CREDITS TO FALLOUT 2

2235: #SOUNDTRACK (GENERAL WASTELAND)- FALLOUT 3

2236: #MADE BY BAILIE-PEIRCE BYRNE (5328)

## SERVER.py:

1: import socket

2: from imports.items import Weapons, Armor, Ammo, Consumables, Misc

3: import threading

4: import time

5: import json

6: import sqlite3

7: import pygame

8: import random

9: import csv

10: import math

11: from pathfinding.core.grid import Grid

12: from pathfinding.finder.a\_star import AStarFinder

13: from imports import constants

14:

15: #Defines the enemies as a sprite group for later reference

16: enemies = pygame.sprite.Group()

17: CLIENTLIST = []

18: CLIENTHISTORY = []

19: connected\_clients = []

20: clientdict = {}

21: player\_positions = {}

22: playermap = {}

23: maps = {}

24: conn = sqlite3.connect('database.db')

25:

26: pingpacket = {"COMMAND":"PNG"}

27: initpacket = {"HP": '0',"X": '0',"Y": '0',"COMMAND":"INIT","map": '0'}

28: playerpacket = {"RECTX": "0","RECTY": "0","COMMAND":"DRAW","ID": "0"}

29: killplayerpacket = {"COMMAND":"KILL","ID":"-2"}

30: enemypacket = {"ID":"0","HP":"1","RECTX":"0","RECTY":"0","NAME":"","COMMAND":"EU","MAP":"-1"}

31:

32: weaponspacket = {"COMMAND" : "WE","I":""}

33: armorpacket = {"COMMAND" : "AR","I":""}

34: ammopacket = {"COMMAND" : "AM","I":""}

35: consumablespacket = {"COMMAND" : "CS","I":""}

36: miscpacket = {"COMMAND" : "MI","I":""}

37:

38: Pweaponspacket = {"COMMAND" : "PWE","I":""}

39: Parmorpacket = {"COMMAND" : "PAR","I":""}

40: Pammopacket = {"COMMAND" : "PAM","I":""}

41: Pconsumablespacket = {"COMMAND" : "PCS","I":""}

42: Pmiscpacket = {"COMMAND" : "PMI","I":""}

43:

44: ContainerLocked = {"COMMAND":"LCK","ID":0}

45: ServerUnlock = {"COMMAND":"SULK","ID":0}

46: QPACKET = {"COMMAND":"QUEST","Q":0,"V":0}

47: QREWARD = {"COMMAND":"REWARD","ITEM":"","SNDR":"","A":0}

48:

49: def read\_csv\_file(csv\_file):

50: with open(constants.mappath + csv\_file, 'r') as file:

51: reader = csv.reader(file)

52: grid = []

53: for row in reader:

54: grid.append([int(cell) for cell in row])

55: maps[csv\_file] = grid

56: return grid

57:

58: read\_csv\_file("test\_PATHING.csv")

59: read\_csv\_file("map2\_PATHING.csv")

60: def sendenemys(id, name, rectx, recty, hp, map):

61: '''

62: Sends all the enemy packets to the clients to accuratley draw the enemies

63: :param id:

64: :param name:

65: :param rectx:

66: :param recty:

67: :param hp:

68: :param map:

69: :return:

70: '''

71: for client in CLIENTLIST:

72: client\_ip = client.getpeername()[0]

73: if playermap[client\_ip] == map:

74: enemypacket["ID"] = id

75: enemypacket["NAME"] = name

76: enemypacket["RECTX"] = rectx

77: enemypacket["RECTY"] = recty

78: enemypacket["HP"] = hp

79: enemypacket["MAP"] = map

80: try:

81: client.send((json.dumps(enemypacket) + "#").encode())

82: except ConnectionResetError: #client disconnected

83: pass

84:

85: class Enemy(pygame.sprite.Sprite):

86: global player\_positions

87: def \_\_init\_\_(self, name, health, speed, damage, x, y, map, id,range):

88: super().\_\_init\_\_()

89: self.id = id

90: self.name = name

91: self.health = health

92: self.speed = speed

93: self.damage = damage

94: self.path = []

95: self.x = int(x)

96: self.y = int(y)

97: self.map = map

98: self.mapdict = {}

99: self.rect = pygame.Rect(self.x, self.y, 32, 32)

100: self.dict = {}

101: self.lastupd = 0

102: self.alive = True

103: self.grid = Grid(matrix=maps["test\_PATHING.csv"])

104: enemies.add(self)

105: self.rect.x = x

106: self.rect.y = y

107: self.range = range

108: self.fleeing = 0

109:

110: def check\_nearest(self):

111: try:

112: if self.alive:

113: closest\_player = None

114: closest\_distance = None

115: enemy\_x = int(self.rect.x)

116: enemy\_y = int(self.rect.y)

117:

118: for player, position in self.dict.items():

119: px, py = map(int, position)

120: distance\_squared = (enemy\_x - px) \*\* 2 + (enemy\_y - py) \*\* 2

121:

122: if closest\_distance is None or distance\_squared < closest\_distance:

123: if str(self.mapdict[player]) == str(self.map):

124: closest\_player = player

125: closest\_distance = distance\_squared

126:

127: if closest\_player is None:

128: self.target = None

129: self.distance = None

130:

131: if closest\_player is not None:

132: self.target = closest\_player

133: self.distance = math.sqrt(closest\_distance)

134: self.targetx, self.targety = map(int, self.dict[closest\_player])

135:

136:

137: if self.distance <= (350\*self.range):

138: start = self.grid.node(self.rect.x // 32, self.rect.y // 32)

139: end = self.grid.node(int(self.targetx // 32), int(self.targety // 32))

140: finder = AStarFinder()

141: self.path, \_ = finder.find\_path(start, end, self.grid)

142: self.grid.cleanup()

143:

144:

145: while len(self.path) > 0:

146: self.move()

147: except RuntimeError:

148: pass

149: def move(self):

150: if time.time() - self.lastupd >= 0.05:

151: next\_pos = self.path[0]

152: if self.rect.x // 32 < next\_pos[0]:

153: self.rect.x += self.speed

154: sendenemys(self.id, self.name, self.rect.x, self.rect.y, self.health, self.map)

155: elif self.rect.x // 32 > next\_pos[0]:

156: self.rect.x -= self.speed

157: sendenemys(self.id, self.name, self.rect.x, self.rect.y, self.health, self.map)

158: if self.rect.y // 32 < next\_pos[1]:

159: self.rect.y += self.speed

160: sendenemys(self.id, self.name, self.rect.x, self.rect.y, self.health, self.map)

161: elif self.rect.y // 32 > next\_pos[1]:

162: self.rect.y -= self.speed

163: sendenemys(self.id, self.name, self.rect.x, self.rect.y, self.health, self.map)

164:

165: try:

166: self.path.pop(0)

167: except IndexError:

168: pass

169: self.lastupd = time.time()

170: @classmethod

171: def from\_csv(cls, name, x, y, map):

172: """

173: :param name:

174: :param x:

175: :param y:

176: :param map:

177: :return: enemy

178: """

179: try:

180: with open(constants.configpath + 'enemies.csv', 'r') as file:

181: reader = csv.DictReader(file)

182: for row in reader:

183: if row['name'] == name:

184: max\_id = max([e.id for e in enemies], default=0)

185: enemy = cls(

186: name=row['name'],

187: health=int(row['health']),

188: speed=int(row['speed']),

189: damage=int(row['damage']),

190: x=x,

191: y=y,

192: map=map,

193: id=max\_id + 1,

194: range= int(row['range'])

195: )

196: print("CREATED")

197: return enemy

198: return None

199: except Exception as e:

200: print(e)

201: def send\_to\_client(ip, client):

202: '''

203: Sends the inital data to the client

204: :param ip:

205: :param client:

206: :return:

207: '''

208: cursor = conn.cursor()

209:

210: # Check if the IP is not present with QID = 0 and insert with COMPLETION set to 0 if not present

211: cursor.execute(

212: """INSERT INTO Player\_Quest\_Data (IP, QUEST, COMPLETION)

213: SELECT ?, Quests.QID, 0

214: FROM Quests

215: WHERE NOT EXISTS (SELECT 1 FROM Player\_Quest\_Data WHERE IP = ? AND QUEST = Quests.QID)""",

216: (ip, ip))

217: # Commit the transaction and close the database connection

218: conn.commit()

219: cursor.execute("SELECT \* FROM Players WHERE IP=?", (ip,))

220: result = cursor.fetchone()

221:

222: if result is None:

223: initpacket["X"] = 0

224: initpacket["Y"] = 0

225: initpacket["HP"] = 100

226: initpacket["map"] = 0

227: else:

228: initpacket["X"] = result[1]

229: initpacket["Y"] = result[2]

230: initpacket["HP"] = result[3]

231: initpacket["map"] = result[6]

232:

233: client.send((json.dumps(initpacket) + "#").encode())

234: def draw\_other\_entities():

235: """

236: Draws other players on other clients

237: :return:

238: """

239: global playerpacket

240: conn = sqlite3.connect('database.db')

241: cursor = conn.cursor()

242: while True:

243: for c in CLIENTLIST:

244: client = c.getpeername()[0]

245: cursor.execute("SELECT \* FROM Players WHERE IP=?", (client,))

246: result = cursor.fetchone()

247: if result is None:

248: pass

249: else:

250: try:

251: x = result[4] # PLAYERRECTX column

252: y = result[5] # PLAYERRECTY column

253: player\_packet = playerpacket.copy()

254: player\_packet["RECTX"] = x

255: player\_packet["RECTY"] = y

256: if client in playermap.keys():

257: pass

258: #map\_key = next(key for key, value in playermap.items() if value == client)

259: except StopIteration:

260: pass

261:

262: if client in clientdict.values():

263: client\_id = next(key for key, value in clientdict.items() if value == client)

264: player\_packet["ID"] = str(client\_id)

265:

266:

267: for c in CLIENTLIST:

268: targetip = c.getpeername()[0]

269: try:

270: if targetip != client and playermap[targetip] == playermap[client]:

271: c.send((json.dumps(player\_packet) + "#").encode())

272: except KeyError: #player ip doesnt exist

273: pass

274: except ConnectionResetError: #player disconnected

275: pass

276: def null\_check():

277: '''

278: Advanced function utilizing aggreagte and concatenating the entirity of the database and scraping through and rectifying null values

279: :return:

280: '''

281: conn = sqlite3.connect('database.db')

282: cursor = conn.cursor()

283:

284: # Fetch all column names in the Players table

285: cursor.execute("PRAGMA table\_info(Players)")

286: columns = [column[1] for column in cursor.fetchall()]

287:

288: # Iterate through each column and update null values with {}

289: for column\_name in columns:

290: update\_query = f"""

291: UPDATE Players

292: SET {column\_name} = '{{}}'

293: WHERE {column\_name} IS NULL

294: """

295: cursor.execute(update\_query)

296:

297: cursor.execute("PRAGMA table\_info(Quests)")

298: columns = [column[1] for column in cursor.fetchall()]

299: for column\_name in columns:

300: update\_query = f"""

301: UPDATE Quests

302: SET {column\_name} = '{0}'

303: WHERE {column\_name} IS NULL

304: """

305: cursor.execute(update\_query)

306: # Commit the changes and close the database connection

307: conn.commit()

308: conn.close()

309: def quest\_update(ip, QID, stage):

310: '''

311: Updated the database with the new completion stage of quests when a player acheieves one.

312: :param ip:

313: :param QID:

314: :param stage:

315: :return:

316: '''

317: try:

318: conn = sqlite3.connect('database.db')

319: cursor = conn.cursor()

320:

321: # Use placeholders (?) for the values to be updated

322: cursor.execute('UPDATE Player\_Quest\_Data SET IP = ?, QUEST = ?, COMPLETION = ? WHERE IP = ? AND QUEST = ?',

323: (ip, QID, stage, ip,QID))

324:

325: # Commit the changes

326: conn.commit()

327: conn.close()

328: except Exception as e:

329: print("Error:", e)

330: def updateinventory(client):

331: '''

332: Resends all clients their inventorys to update the clients as much as possible without exausting the server connection.

333: :param client:

334: :return:

335: '''

336: conn = sqlite3.connect('database.db')

337: cursor = conn.cursor()

338: cursor.execute("SELECT \* FROM Players WHERE IP=?", (client.getpeername()[0],))

339: result = cursor.fetchone()

340: try:

341: Pweaponspacket["I"] = result[7]

342: Parmorpacket["I"] = result[8]

343: Pammopacket["I"] = result[9]

344: Pconsumablespacket["I"] = result[10]

345: Pmiscpacket["I"] = result[11]

346:

347: client.send((json.dumps(Pweaponspacket) + "#").encode())

348: client.send((json.dumps(Parmorpacket) + "#").encode())

349: client.send((json.dumps(Pammopacket) + "#").encode())

350: client.send((json.dumps(Pconsumablespacket) + "#").encode())

351: client.send((json.dumps(Pmiscpacket) + "#").encode())

352:

353: cursor.execute('SELECT QUEST, COMPLETION FROM Player\_Quest\_Data WHERE IP = ?', (client.getpeername()[0],))

354: records = cursor.fetchall()

355:

356: for record in records:

357: QPACKET["Q"] = record[0]

358: QPACKET["V"] = record[1]

359: client.send((json.dumps(QPACKET) + "#").encode())

360:

361: cursor.execute('SELECT ID FROM Containers WHERE Locked = 1')

362: records = cursor.fetchall()

363:

364: for record in records:

365: ContainerLocked["ID"] = record[0]

366: client.send((json.dumps(ContainerLocked) + "#").encode())

367:

368:

369: except:

370: pass

371: conn.close()

372: def remove\_item(ip, item\_name, amount\_to\_remove):

373: '''

374: Removes items from inventorys

375: :param ip:

376: :param item\_name:

377: :param amount\_to\_remove:

378: :return:

379: '''

380: try:

381: conn = sqlite3.connect('database.db')

382: cursor = conn.cursor()

383:

384: # Retrieve the current row for the specified IP

385: cursor.execute("SELECT Weapons, Armor, Ammo, Consumables, Misc FROM Players WHERE IP = ?", (ip,))

386: player\_data = cursor.fetchone()

387:

388: if player\_data is not None:

389: weapons, armor, ammo, consumables, misc = player\_data

390:

391: # Convert the fields from JSON to Python dictionaries

392: weapons = json.loads(weapons) if weapons else {}

393: armor = json.loads(armor) if armor else {}

394: ammo = json.loads(ammo) if ammo else {}

395: consumables = json.loads(consumables) if consumables else {}

396: misc = json.loads(misc) if misc else {}

397:

398: # Check if the item exists in any of the fields and update its amount

399: fields = {'Weapons': weapons, 'Armor': armor, 'Ammo': ammo, 'Consumables': consumables, 'Misc': misc}

400: for field\_name, field\_data in fields.items():

401: if item\_name in field\_data:

402: current\_amount = field\_data[item\_name]

403: if current\_amount >= amount\_to\_remove:

404: field\_data[item\_name] -= amount\_to\_remove

405: if field\_data[item\_name] == 0:

406: del field\_data[item\_name] # Delete the key if the new value is 0

407: else:

408: print(f"Insufficient {item\_name} in {field\_name} for IP '{ip}'.")

409: remove\_item(ip,item\_name,current\_amount)

410: return

411:

412: # Stop checking other fields once the item is found and updated

413:

414: # Convert the modified dictionaries back to JSON

415: weapons\_json = json.dumps(weapons)

416: armor\_json = json.dumps(armor)

417: ammo\_json = json.dumps(ammo)

418: consumables\_json = json.dumps(consumables)

419: misc\_json = json.dumps(misc)

420:

421: # Update the database with the new field values

422: cursor.execute("UPDATE Players SET Weapons = ?, Armor = ?, Ammo = ?, Consumables = ?, Misc = ? WHERE IP = ?",

423: (weapons\_json, armor\_json, ammo\_json, consumables\_json, misc\_json, ip))

424: conn.commit()

425: print(f"Removed {amount\_to\_remove} {item\_name}(s) from the player's inventory for IP '{ip}'.")

426:

427: else:

428: print(f"Player with IP '{ip}' not found in the database.")

429:

430: except sqlite3.Error as e:

431: print("SQLite error:", e)

432:

433: finally:

434: # Close the database connection

435: if conn:

436: conn.close()

437: def add\_item(ip, item\_name, amount\_to\_add, field\_name):

438: '''

439: Opposite of remove , instead adds items to speicidied fields

440: :param ip:

441: :param item\_name:

442: :param amount\_to\_add:

443: :param field\_name:

444: :return:

445: '''

446: try:

447: conn = sqlite3.connect('database.db')

448: cursor = conn.cursor()

449:

450: # Retrieve the current row for the specified IP

451: cursor.execute("SELECT " + field\_name + " FROM Players WHERE IP = ?", (ip,))

452: player\_data = cursor.fetchone()

453:

454: if player\_data is not None:

455: field\_data = player\_data[0]

456: item\_dictionary = json.loads(field\_data) if field\_data else {}

457:

458: if item\_name in item\_dictionary:

459: item\_dictionary[item\_name] += amount\_to\_add

460: else:

461: item\_dictionary[item\_name] = amount\_to\_add

462:

463: # Convert the modified dictionary back to JSON

464: updated\_field\_json = json.dumps(item\_dictionary)

465:

466: # Update the database with the new field value

467: cursor.execute("UPDATE Players SET " + field\_name + " = ? WHERE IP = ?", (updated\_field\_json, ip))

468: conn.commit()

469: print(f"Added {amount\_to\_add} {item\_name}(s) to the player's inventory for IP '{ip}' in the {field\_name} field.")

470:

471: else:

472: print(f"Player with IP '{ip}' not found in the database.")

473:

474: except sqlite3.Error as e:

475: print("SQLite error:", e)

476:

477: finally:

478: # Close the database connection

479: if conn:

480: conn.close()

481: def update\_db(client):

482: #main packet handler

483: """

484: Major function handling all of the packets into the server, these are threads created for each client so each client has their own function stack for improved efficiency.

485: :param client:

486: :return:

487: """

488: global player\_positions

489: conn = sqlite3.connect('database.db')

490: cursor = conn.cursor()

491: while True:

492: try:

493: data = client.recv(2048)

494: except ConnectionResetError:

495: ping\_client(client)

496: return

497: if not data:

498: continue

499: data\_list = data.decode().split("#")

500: for msg\_str in data\_list:

501: if msg\_str != "":

502: try:

503: msg = json.loads(msg\_str)

504: except:

505: pass

506: try:

507: if msg["COMMAND"] == "INITINV":

508: updateinventory(client)

509: if msg["COMMAND"] == "GENERAL":

510: ip = msg["IP"]

511: x = msg["X"]

512: y = msg["Y"]

513: hp = msg["HP"]

514: rectx = msg["RECTX"]

515: recty = msg["RECTY"]

516: map = msg["Map"]

517: cursor.execute("SELECT \* FROM Players WHERE IP=?", (ip,))

518: result = cursor.fetchone()

519: player\_positions[ip] = (rectx, recty)

520: playermap[ip] = map

521: if result is None:

522: cursor.execute("INSERT INTO Players (IP, X, Y, HP, PLAYERRECTX, PLAYERRECTY, 'Map') VALUES (?, ?, ?, ?, ?, ?, ?)", (ip, x, y, hp, rectx, recty, map))

523: conn.commit()

524: null\_check()

525: else:

526: cursor.execute("UPDATE Players SET X=?, Y=?, HP=?, PLAYERRECTX=?, PLAYERRECTY=? , 'Map'=? WHERE IP=?", (x, y, hp, rectx, recty, map ,ip))

527: conn.commit()

528:

529: #for client in client list - if client != conn then send the packet to client

530: if msg["COMMAND"] == "RMV":

531: remove\_item(client.getpeername()[0],msg["AMMO"],msg["AMOUNT"])

532: if msg["COMMAND"] == "EH": #Enemy Hit

533: eid = msg["ID"]

534: dmg = msg["DMG"]

535: for enemy in enemies:

536: if enemy.id == eid:

537: enemy.health -= dmg

538: try:

539: enemy.targetx = int(player\_positions[(client.getpeername()[0])][0])

540: enemy.targety = int(player\_positions[(client.getpeername()[0])][1])

541: enemy.distance = 0

542: except Exception as e:

543: print (e)

544: if enemy.health <= 0:

545: enemy.alive = False

546: sendenemys(enemy.id,enemy.name, enemy.rect.x,enemy.rect.y, enemy.health,enemy.map)

547: enemies.remove(enemy)

548: amount = random.randint(5,100)

549: add\_item(client.getpeername()[0],"Cap(s)",amount,"Misc")

550: QREWARD["ITEM"] = "Cap(s)"

551: QREWARD["A"] = amount

552: QREWARD["SNDR"] = enemy.name

553: client.send((json.dumps(QREWARD) + "#").encode())

554: updateinventory(client)

555: if eid not in enemies:

556: sendenemys(eid, 0, 0, 0, 0, 0)

557: if msg["COMMAND"] == "OC":# OPEN CONTAINER

558: ContID = msg["ID"]#GET CONTAINER ID FROM REQUEST

559: locked = False

560: cursor.execute("SELECT ID FROM Containers WHERE ID = ? AND Locked = 1",(ContID,))

561:

562: if cursor.fetchone() != None: #Checks if container is locked

563: locked = True

564:

565: query = "SELECT Weapons, Armor, Ammo, Consumables, Misc FROM Containers WHERE ID = ?" #GET CONTENT FROM DB

566: cursor.execute(query, (ContID,))

567: container\_data = cursor.fetchone()

568:

569: #SERIALIZE THE DATA INTO JSON DICTIONARIES FOR SENDING

570:

571: weapons = container\_data[0]

572: armor = container\_data[1]

573: ammo = container\_data[2]

574: consumables = container\_data[3]

575: misc = container\_data[4]

576:

577: weaponspacket["I"] = weapons

578: armorpacket["I"] = armor

579: ammopacket["I"] = ammo

580: consumablespacket["I"] = consumables

581: miscpacket["I"] = misc

582:

583: #SEND PACKETS INDIVIDUALLY TO REDUCE CHANCE OF BUFFER OVERFLOW ON CLIENT

584: client.send((json.dumps(weaponspacket) + "#").encode())

585: client.send((json.dumps(armorpacket) + "#").encode())

586: client.send((json.dumps(ammopacket) + "#").encode())

587: client.send((json.dumps(consumablespacket) + "#").encode())

588: client.send((json.dumps(miscpacket) + "#").encode())

589: # ADD THE SQL NULL CHECKS HERE AT THE START OF THE FC AND TC CHECKS.

590: if msg["COMMAND"] == "TC":

591: ContID = msg["ID"]

592: Tab = msg["TAB"]

593: Item = msg["Item"]

594: Amount = int(msg["A"]) # Convert amount to an integer

595: client\_ip = client.getpeername()[0]

596: # Fetch container data

597: container\_query = f"SELECT {Tab} FROM Containers WHERE ID = ?"

598: cursor.execute(container\_query, (ContID,))

599: container\_data = json.loads(cursor.fetchone()[0])

600:

601: # Fetch player's inventory data

602: player\_query = f"SELECT {Tab} FROM Players WHERE IP = ?"

603: cursor.execute(player\_query, (client\_ip,))

604: fetch\_result = cursor.fetchone()

605: if fetch\_result is not None:

606: player\_data = json.loads(fetch\_result[0])

607: else:

608: player\_data = {}

609:

610: # Check if the item exists in player's inventory and if there's enough

611: if Item in player\_data and player\_data[Item] >= Amount:

612: # Update container data

613: if Item in container\_data:

614: container\_data[Item] += Amount

615: else:

616: container\_data[Item] = Amount

617:

618: # Update player's inventory data

619: player\_data[Item] -= Amount

620:

621: # Remove item from player's inventory and container if its amount becomes zero

622: if container\_data[Item] <= 0:

623: del container\_data[Item]

624:

625: if player\_data[Item] <= 0:

626: del player\_data[Item]

627:

628: # Update container data in the database

629: container\_query = f"UPDATE Containers SET {Tab} = ? WHERE ID = ?"

630: updated\_container = json.dumps(container\_data)

631: cursor.execute(container\_query, (updated\_container, ContID))

632:

633: # Update player's inventory data in the database

634: player\_query = f"UPDATE Players SET {Tab} = ? WHERE IP = ?"

635: updated\_inventory = json.dumps(player\_data)

636: cursor.execute(player\_query, (updated\_inventory, client\_ip))

637:

638: # Commit changes

639: conn.commit()

640: else:

641: print("Item not found in player's inventory or insufficient amount.")

642: if msg["COMMAND"] == "FC":

643: ContID = msg["ID"]

644: Tab = msg["TAB"]

645: Item = msg["Item"]

646: Amount = int(msg["A"]) # Convert amount to an integer

647: client\_ip = client.getpeername()[0]

648:

649: # Fetch container data

650: container\_query = f"SELECT {Tab} FROM Containers WHERE ID = ?"

651: cursor.execute(container\_query, (ContID,))

652: container\_data = json.loads(cursor.fetchone()[0])

653:

654: # Fetch player's inventory data

655: player\_query = f"SELECT {Tab} FROM Players WHERE IP = ?"

656: cursor.execute(player\_query, (client\_ip,))

657: fetch\_result = cursor.fetchone()

658: if fetch\_result is not None:

659: player\_data = json.loads(fetch\_result[0])

660: else:

661: player\_data = {}

662:

663: # Check if the item exists in container and if there's enough

664: if Item in container\_data and container\_data[Item] >= Amount:

665: # Update player's inventory data

666: if Item in player\_data:

667: player\_data[Item] += Amount

668: else:

669: player\_data[Item] = Amount

670:

671: # Update container data

672: container\_data[Item] -= Amount

673:

674: # Remove item from player's inventory and container if its amount becomes zero

675: if container\_data[Item] <= 0:

676: del container\_data[Item]

677:

678: if player\_data[Item] <= 0:

679: del player\_data[Item]

680:

681: # Update player's inventory data in the database

682: player\_query = f"UPDATE Players SET {Tab} = ? WHERE IP = ?"

683: updated\_inventory = json.dumps(player\_data)

684: cursor.execute(player\_query, (updated\_inventory, client\_ip))

685:

686: # Update container data in the database

687: container\_query = f"UPDATE Containers SET {Tab} = ? WHERE ID = ?"

688: updated\_container = json.dumps(container\_data)

689: cursor.execute(container\_query, (updated\_container, ContID))

690:

691: # Commit changes

692: conn.commit()

693: else:

694: print("Item not found in container or insufficient amount.")

695: if msg["COMMAND"] == "QU":

696: print("TRYING QUEST UPDATE")

697: quest\_update(client.getpeername()[0],int(msg["ID"]),int(msg["VALUE"]))

698: if int(msg["ID"]) == 0 and int(msg["VALUE"]) == 1:

699: remove\_item(client.getpeername()[0],"Key",1)

700: print("Item Removed")

701: add\_item(client.getpeername()[0],"Medkit",3,"Consumables")

702: QREWARD["ITEM"] = "Medkit"

703: QREWARD["A"] = 3

704: QREWARD["SNDR"] = "Tony"

705: client.send((json.dumps(QREWARD) + "#").encode())

706: updateinventory(client)

707: if int(msg["ID"]) == 999 and int(msg["VALUE"]) == 1:

708: add\_item(client.getpeername()[0],"Trophy",1,"Misc")

709: QREWARD["ITEM"] = "Trophy"

710: QREWARD["A"] = 1

711: QREWARD["SNDR"] = "Shade"

712: client.send((json.dumps(QREWARD) + "#").encode())

713: updateinventory(client)

714: if int(msg["ID"]) == 1 and int(msg["VALUE"]) == 1:

715: remove\_item(client.getpeername()[0], "Machete", 1)

716: add\_item(client.getpeername()[0], "Shotgun", 1, "Weapons")

717: add\_item(client.getpeername()[0], "Shotgun Shells", 25, "Ammo")

718: QREWARD["ITEM"] = "Shotgun and Shells"

719: QREWARD["A"] = 1

720: QREWARD["SNDR"] = "Wrangler"

721: client.send((json.dumps(QREWARD) + "#").encode())

722: updateinventory(client)

723: if msg["COMMAND"] == "ULK":

724: cursor.execute("UPDATE Containers SET Locked = 0 WHERE ID = ?", (int(msg["ID"]),))

725: conn.commit()

726: updateinventory(client)

727: for client in CLIENTLIST:

728: client.send((json.dumps(ServerUnlock) + "#").encode())

729: except ConnectionResetError:

730: ping\_client(client)

731: return

732: def ping\_client(client):

733: """

734: Attempts to send ping packet to client if unsuccsesful removes client from connected clients and kills their player sprite

735: :param client,, type is socket connection:

736: :return:

737: """

738: try:

739: client.send((json.dumps(pingpacket)+"#").encode())

740: except:

741: print(client.getpeername()[0]," Disconnected")

742: try:

743: CLIENTLIST.remove(client)

744: player\_positions.pop(client.getpeername()[0])

745: connected\_clients.remove(client.getpeername()[0])

746: except ValueError:

747: #client has already been removed from conneections lists

748: pass

749: except KeyError:

750: pass

751: killplayerpacket["ID"] = (list(clientdict.keys())[list(clientdict.values()).index(client.getpeername()[0])])

752: print("KILLING SPRITE ", (list(clientdict.keys())[list(clientdict.values()).index(client.getpeername()[0])]))

753: try:

754: for client in CLIENTLIST:

755: client.send((json.dumps(killplayerpacket)+"#").encode())

756: killplayerpacket["ID"] = "-2"

757: except:

758: pass

759: return

760: def manage\_connections():

761: """

762: Calls the ping\_client string

763: Handles Pinging connected socket connections to ensure their availbility if not removes them from the Connected client string

764: :return:

765: """

766: while True:

767: global CLIENTLIST

768: for client in CLIENTLIST:

769: ping\_client(client)

770: time.sleep(5)

771: def enemyhandler():

772: '''

773: Thread used to continously update and handle all enemy instances

774: :return:

775: '''

776: ForceEnemyUpdate = 0

777: while True:

778: global enemies

779: for e in enemies:

780: if e.alive:

781: e.dict = player\_positions

782: e.mapdict = playermap

783: if time.time() - ForceEnemyUpdate >= 0.5:

784: try:

785: sendenemys(e.id, e.name, e.rect.x, e.rect.y, e.health, e.map)

786: except KeyError:

787: pass

788: ForceEnemyUpdate = time.time()

789: e.check\_nearest()

790: else:

791: pass

792: def spawn\_enemies():

793: '''

794: Spawns all enemies

795: :return:

796: '''

797: with open(constants.configpath + 'enemyspawns.csv', 'r') as file:

798: reader = csv.DictReader(file)

799: # Loop through each row in the file

800: for row in reader:

801: mapid = str(row["mapID"])

802: name = str(row["name"])

803: rectx = int(row['rectx'])

804: recty = int(row['recty'])

805: Enemy.from\_csv(name,rectx,recty,mapid)

806: print("active threads ", threading.active\_count())

807:

808:

809: # Return None if we couldn't find an enemy with the given name

810: return None

811: def saturatecontainers():

812: '''

813: Saturates containers using hardcoded values and random spontaneity

814: Some items are black listed

815: :return:

816: '''

817: conn = sqlite3.connect('database.db')

818: cursor = conn.cursor()

819:

820: # Access the Containers table

821: cursor.execute("SELECT ID, Weapons, Armor, Ammo, Consumables, Misc FROM Containers")

822: containers\_data = cursor.fetchall()

823:

824: # Create a dictionary to map category names to the corresponding item dictionaries

825: categories = {

826: "Weapons": Weapons,

827: "Armor": Armor,

828: "Ammo": Ammo,

829: "Consumables": Consumables,

830: "Misc": Misc

831: }

832:

833: # Customize the quantity ranges for different item categories

834: def get\_quantity\_range(category, container\_id):

835: range = quantity\_ranges.get(category)

836: if container\_id == 0:

837: return range

838: else:

839: return (range[0], range[1] \* container\_id)

840:

841: quantity\_ranges = {

842: "Weapons": (1, 1),

843: "Ammo": (5, 15),

844: "Armor": (1,1),

845: "Consumables": (1, 2),

846: "Misc": (1, 2)

847: }

848:

849: # Iterate through container data and update the item dictionary

850: for container\_data in containers\_data:

851: container\_id = container\_data[0]

852: category\_data = container\_data[1:]

853:

854: for category, item\_data\_json in zip(categories.keys(), category\_data):

855: if item\_data\_json:

856: item\_data = json.loads(item\_data\_json)

857:

858: item\_dict = categories.get(category)

859:

860: if item\_dict is not None: #Black list items from occuring naturally here

861: if category == "Ammo" and None in item\_dict:

862: item\_dict.pop(None)

863: if category == "Misc" and "Trophy" in item\_dict:

864: item\_dict.pop("Trophy")

865:

866: # Choose a random item from the category

867: random\_item = random.choice([item for item in item\_dict.keys() if item != "None"])

868:

869: # Randomly select a quantity based on the customized range

870: min\_quantity, max\_quantity = get\_quantity\_range(category, container\_id)

871: random\_quantity = random.randint(min\_quantity, max\_quantity)

872:

873: # Check if the item exists in the container's item dictionary

874: if random\_item in item\_data:

875: item\_data[random\_item] += random\_quantity

876: else:

877: item\_data[random\_item] = random\_quantity

878:

879: # Update the container's corresponding column with the updated item dictionary

880: updated\_item\_data\_json = json.dumps(item\_data)

881: cursor.execute(f"UPDATE Containers SET {category} = ? WHERE ID = ?",

882: (updated\_item\_data\_json, container\_id))

883:

884:

885: # Commit the changes and close the connection

886: conn.commit()

887: conn.close()

888: def lockcontainers():

889: '''

890: Randomly locks containers based on their ID so earlier containers are less likeley to be locked

891: :return:

892: '''

893: conn = sqlite3.connect("database.db")

894: cursor = conn.cursor()

895:

896: # Access the Containers table

897: cursor.execute("SELECT ID FROM Containers WHERE ID != 0")

898: container\_ids = [row[0] for row in cursor.fetchall()]

899:

900: for container\_id in container\_ids:

901: # Determine the lock status based on the container ID

902: lock\_chance = container\_id \* 10 # ID0 is always unlocked, ID1 is 10% chance

903:

904: if lock\_chance > 75 :

905: lock\_chance = 75

906:

907: # Generate a random number between 1 and 100

908: random\_number = random.randint(1, 100)

909:

910: if random\_number <= lock\_chance:

911: # Set the container as locked (1) if the random number is less than or equal to the lock chance

912: cursor.execute("UPDATE Containers SET Locked = 1 WHERE ID = ?", (container\_id,))

913: else:

914: # Set the container as unlocked (0) otherwise

915: cursor.execute("UPDATE Containers SET Locked = 0 WHERE ID = ?", (container\_id,))

916:

917: # Commit the changes and close the connection

918: conn.commit()

919: conn.close()

920:

921: saturatecontainers()

922: lockcontainers()

923: #MAin block to accept connections and add IPs and CONN\_SOCKS to the correct lists.

924: with socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) as s:

925: s.bind((constants.IP\_ADDRESS,constants.PORT))

926: s.listen()

927: print("Server is listening....")

928: lock = threading.Lock()

929: threading.Thread(target=manage\_connections, args=()).start()

930: threading.Thread(target=draw\_other\_entities, args=()).start()

931: spawn\_enemies()

932: threading.Thread(target=enemyhandler, args=()).start()

933:

934: while True:

935: conn\_sock, address = s.accept()

936: print(f"Client {address[0]} connected")

937: send\_to\_client(address[0],conn\_sock)

938: null\_check()

939: if address[0] not in connected\_clients:

940: CLIENTLIST.append(conn\_sock)

941: connected\_clients.append(address[0])

942: CLIENTHISTORY.append(address[0])

943: for c, key in enumerate(CLIENTHISTORY):

944: clientdict[c] = key

945: null\_check()

946: threading.Thread(target=update\_db, args=(conn\_sock,)).start()

### Constants.py:

1: import ctypes

2: user32 = ctypes.windll.user32

3: IP\_ADDRESS = "192.168.0.117" #this is subject to change depending on the host

4: PORT = 60002

5: assetpath = "assets/"

6: mappath = "maps/"

7: configpath = "config/"

8: g = 9.81

9:

10: screen\_width = int(user32.GetSystemMetrics(0))

11: screen\_height = int(user32.GetSystemMetrics(1))

12:

13: menu\_width = screen\_width

14: menu\_height = screen\_height

15: menu\_x = (screen\_width - menu\_width) // 2

16: menu\_y = (screen\_height - menu\_height) // 2

17:

18: equipped\_x = menu\_x + menu\_width - 50

19: equipped\_y = (menu\_y + menu\_height) // 2 - (40) \* 2

20: equipped\_size = 30

21: equipped\_spacing = 10

22:

23:

24:

25: PLAYER\_INVENTORY\_POS = (20, 60)

26: CONTAINER\_INVENTORY\_POS = (screen\_width // 2 + 20, 60) # Adjusted position for the right side

27: TAB\_WIDTH = 130

28: TAB\_HEIGHT = 30

29: FONT\_SIZE = 20

30: WHITE = (255, 255, 255)

31: GRAY = (150, 150, 150)

32: DARK\_GRAY = (50, 50, 50)

33: BLUE = (0, 0, 255)

34: GREEN = (0, 255, 0)

35: BLACK = (0, 0, 0)

36: RED = (255,0,0)

### Items.py:

1: # CATEGORYS ARE DICTIONARIES//

2: # WEAPONS INDEXED AS Weapons["ITEM"][X]// (X) 0 =DMG, 1 = FIRE-RATE, 2 = AMMO\_TYPE

3: # ARMOR IS INDEXED AS Armor["ITEM"][X]// (X) 0 =SLOT, 1 = DAMAGE REDUCTION

4: # AMMO IS INDEXED AS Ammo["ITEM"][X]// (X) 0 = SPEED, 1= PENETRATION VALUE

5: # CONSUMABLES IS INDEXED AS Consumables["ITEM"][X]// (X) 0 = EFFECT, 1 =AMOUNT

6: # MISC IS INDEXED AS Misc["ITEM"][X]// (X) 0 = Description

7:

8: Weapons = {

9: "SMG": ("smg.png", 5, 0.1, "9mm Ammo"),

10: "Bat": ("bat.png", 2, 5, None),

11: "Pistol": ("pistol.png", 1, 1, "9mm Ammo"),

12: "Machete": ("machete.png", 8, 3,None),

13: "Shotgun": ("shotgun.png", 10, 1.5, "Shotgun Shells"),

14: "Rifle": ("rifle.png", 7, 0.2, "5.56mm Ammo"),

15: "Sniper Rifle": ("heavysnipe.png", 10, 2.0, "50cal Ammo")

16: }

17: Armor = {

18: "Hat": ("hat.png", "H", 1),

19: "Crocs": ("crocs.png", "F", 1),

20: "Shirt": ("shirt.png", "C", 1),

21: "Combat Helmet": ("helmet.png", "H", 5),

22: "Combat Chestplate": ("cchest.png", "C", 10),

23: "Combat Boots": ("cboots.png", "F", 5),

24: "T-45 Power Armor Helmet": ("t45h.png", "H", 15),

25: "T-45 Power Armor Chestplate": ("t45c.png", "C", 30),

26: "T-45 Power Armor Boots": ("t45f.png", "F", 15)

27: }

28: Ammo = {

29: None: ("9mm.png", 0, None),

30: "9mm Ammo": ("9mm.png", 1, 1),

31: "50cal Ammo": ("50cal.png", 4, 10),

32: "Shotgun Shells": ("shells.png", 2, 1),

33: "5.56mm Ammo": ("5.56mm.png", 2, 2)

34: }

35: Consumables = {

36: "Medkit": ("medkit.png", "HEAL", 50),

37: "Bandages": ("bandages.png", "HEAL", 15),

38: "Plasters": ("plasters.png", "HEAL", 5),

39: "Survival Kit": ("kit.png", "HEAL", 100),

40: "Coca-Cola": ("coke.png", "HEAL", 3)

41: }

42: Misc = {"Key": ("key.png",("A Rusty Key With The Word Tony Engraved")), "Lockpick": ("key.png",("Simple Lockpick")), "Cap(s)":("cap.png",("Currency Of Some Sort")),"Trophy":("trophy.png",("A Trophy From Shade To Show You Couldve Left The Wasteland But Stayed"))}

### Startup.py:

1: import os

2: import sqlite3

3: import pip

4: pip.main(['install', 'pytmx==3.31'])

5:

6:

7: def create\_tables():

8: # Check if the database file exists

9: if os.path.isfile("database.db"):

10: print("Database file exists.")

11: # Connect to the database

12: with sqlite3.connect("database.db") as conn:

13: cursor = conn.cursor()

14:

15: # Check if the Players table exists

16: cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='Players'")

17: if cursor.fetchone():

18: print("Players table successfully verified.")

19: else:

20: # Create the Players table

21: cursor.execute("""

22: CREATE TABLE Players (

23: IP TEXT,

24: X INT,

25: Y INT,

26: HP INT,

27: PLAYERRECTX INT,

28: PLAYERRECTY INT,

29: Map TEXT,

30: Weapons TEXT,

31: Armor TEXT,

32: Ammo TEXT,

33: Consumables TEXT,

34: Misc TEXT

35: )

36: """)

37: print("Players table created.")

38:

39: # Check if the Containers table exists

40: cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='Containers'")

41: if cursor.fetchone():

42: print("Containers table successfully verified.")

43: else:

44: # Create the Containers table

45: cursor.execute("""

46: CREATE TABLE Containers (

47: ID INTEGER PRIMARY KEY,

48: Weapons TEXT NOT NULL,

49: Armor TEXT NOT NULL,

50: Ammo TEXT NOT NULL,

51: Consumables TEXT NOT NULL,

52: Misc TEXT NOT NULL,

53: Locked INTEGER

54: )

55: """)

56: print("Containers table created.")

57:

58: # Insert 10 entries with IDs from 0 to 10 with blank inventory spaces.

59: entries = [(i,"{}","{}","{}","{}","{}", 0) for i in range(11)]

60: cursor.executemany("INSERT INTO Containers (ID, Weapons, Armor, Ammo, Consumables, Misc, Locked) VALUES (?, ?, ?, ?, ?, ?, ?)",entries)

61: print("Inserted 10 entries with IDs from 0 to 10 and default values.")

62:

63: # Check if the Quests table exists

64: cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='Quests'")

65: if cursor.fetchone():

66: print("Quests table successfully verified.")

67: else:

68: # Create the Quests table

69: cursor.execute("""

70: CREATE TABLE Quests (

71: QID INTEGER,

72: MAX INTEGER

73: )

74: """)

75: entries = [(i,0) for i in range(5)]

76: cursor.executemany("INSERT INTO Quests (QID, MAX) VALUES (?, ?)",entries) # add all the quests ids

77: cursor.execute("INSERT INTO Quests (QID, MAX) VALUES (999, 1)") #add the final quest

78:

79: print("Quests table created.")

80:

81: # Check if the Player\_Quest\_Data table exists

82: cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='Player\_Quest\_Data'")

83: if cursor.fetchone():

84: print("Player\_Quest\_Data table successfully verified.")

85: else:

86: # Create the Player\_Quest\_Data table

87: cursor.execute("""

88: CREATE TABLE Player\_Quest\_Data (

89: IP INTEGER NOT NULL,

90: QUEST INTEGER,

91: COMPLETION INTEGER

92: )

93: """)

94: print("Player\_Quest\_Data table created.")

95:

96: # Check if the login\_info table exists

97: cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='login\_info'")

98: if cursor.fetchone():

99: print("login\_info table successfully verified.")

100: else:

101: # Create the login\_info table

102: cursor.execute("""

103: CREATE TABLE login\_info (

104: IP INTEGER,

105: Username TEXT,

106: Password TEXT

107: )

108: """)

109: print("login\_info table created.")

110:

111: print("Tables checked and created if necessary.")

112: print("Database Operational.")

113:

114: else:

115: print("Database file does not exist.")

116: with open("database.db", "w") as database\_file:

117: print("Database file created.")

118: database\_file.close()

119: create\_tables()

120:

121:

122: create\_tables()

123:

124: #Self explanatory try except catch to attempt to host a server and provide an error to the user.

125: try:

126: exec(open('SERVER.py').read())

127: except:

128: print("Server File not found and or is damaged please ensure SERVER.py is present in the same directory as this file")

129: print("Or try Reinstalling the SERVER.py")