“Dangeresque 2” Design Docs

Contributors:

Anthony, Ariel, Briar, Greyson

Interactable Objects

Coding: Greyson

Testing: Briar

Overview

- Items within the map can be interacted with
- All items not part of the background can be looked at and talked to
- Some items can be used
- Some items can be added to the player's inventory

Technical

- InteractableObjects can be talked with & looked at with customized strings and dialogue flows per item
- UseableObjects get passed a custom "use" function within their constructor that gets called when used
- GettableObjects have an additional "get" method that adds the item to the inventory and removes it from visibility
- InteractableObjects inherit pygame sprite, UseableObjects inherit InteractableObjects, and GettableObjects inherit UseableObjects

Main Character, Movement, & Collision

Coding: Ariel

Testing: Briar

Overview

- Ability to move the character using WASD keys and Arrow keys.
- Ability to be able to sprint and move faster
- Utilize the player location in order to add collision and proper coordinate for rotation
- Show different animation for every different type of direction with the movement
- Show the proper animation when the map rotates
- Add special events for certain collisions.

Technical:

- Utilize a player class to add the sprite to character
- Utilize the rect method from pygame to add a rectangle to the sprite that can be moved
- Record the old and new positions of the characters to make collision stop the character when it come to contact with it or play a special event
- Utilize the old and new position of the new character to rotate it depending on the direction of the map

Main Menu

Coding: Ariel

Testing: Anthony

Overview:

- Added a menu that has the option to create a new game (start the game) or quit (close pygame)
- Added a loading image before the main menu class closes to simulate a “loading screen”
- Currently in the process of adding an option menu with user interactable accessibility settings
- Currently in the process of adding a load game button which will use the saving/ loading state class.

Technical:

- Utilized a button class that can be used to create buttons and also resize those buttons and control the location of the buttons.
- Utilized rect.collide to change the sprite of the button to make it appear interactable
- Utilized “states” in order to control what is being displayed on the screen whenever the player press button

Saving & Loading the Game State

Coding: Anthony

Testing: Greyson

Overview

- Ability to Load and Save Game State
- Create a Save File on Initial Launch using Default Object Data
- Automatically Save File on Game Exit
- Keep history of last 3 saves, delete the extras

Technical

- Use Serialized JSON data stored in a data file (data.txt)
- Object states are stored and interacted with within game code in JSON Format.
- When Loaded, a Custom Deserializer is used to take save files and make it into "readable" JSON data for the game to restore states from.
- When Saved, a Custom Serializer is used to store the current game state into the save file to be loaded upon relaunch of the game.
- Save files will contain a timestamp of when they were saved. When the game is loaded, the program will load the latest save file and will create a new one with the current timestamp that will hold any modifications made during the game's execution. If more than 3 save files exist, the oldest one will be deleted. When the game is saved the filename of the current save file will be updated to reflect the timestamp it was saved.

Drawing Maps & Rotation

Coding: Briar

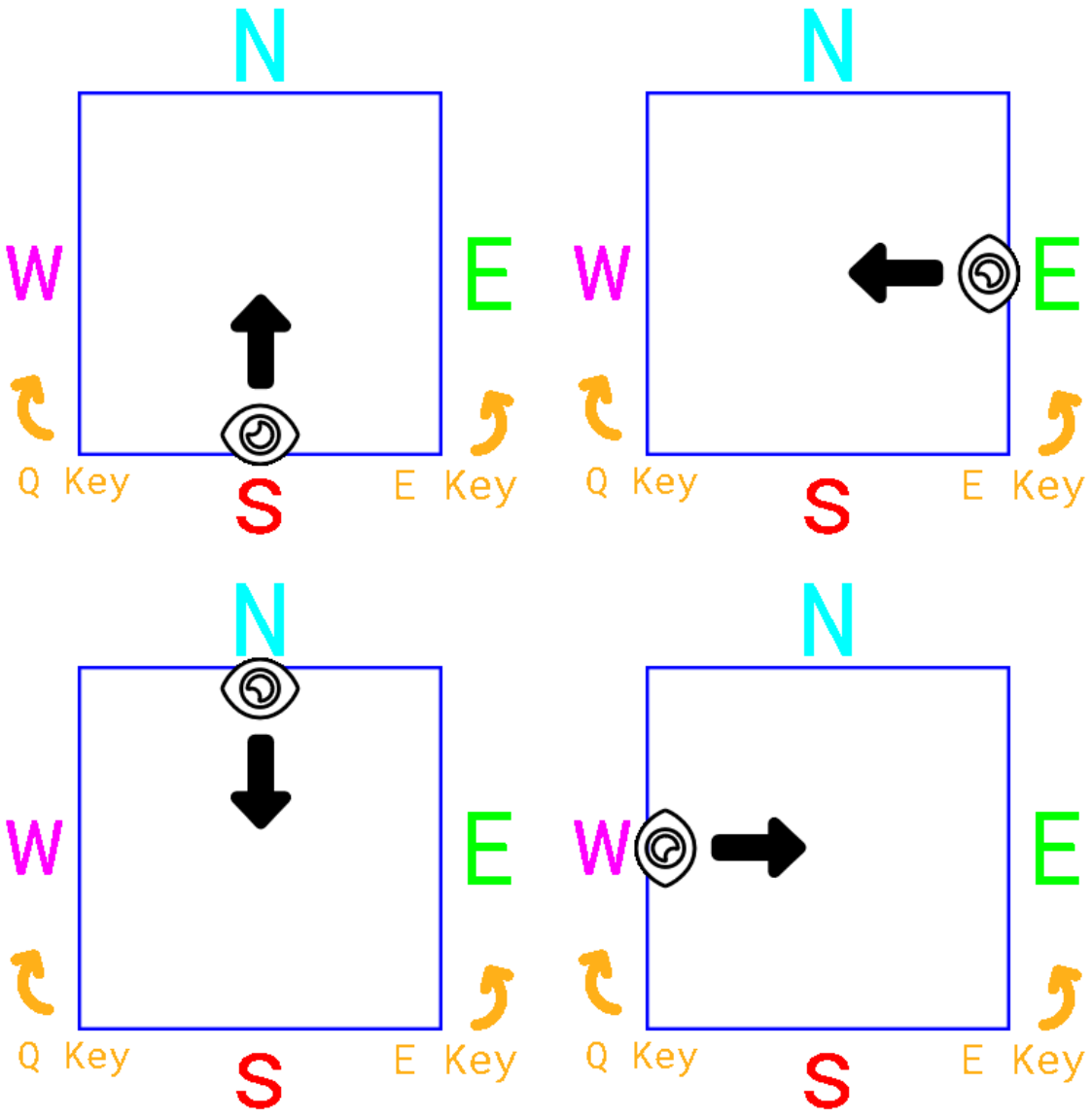
Testing: Ariel

Overview

Working within our chosen library's limitations, we decided that having multiple views per room to simulate a 3d room would bring a fresh perspective to the point and click gameplay.

Goals

- Allow the end user to rotate between four different views of each major room, pressing Q and E to turn to face the left and right walls in relation to the current view of the room.



- Clearly indicate to the user with UI elements whether or not they can rotate the view of the current location, and what the current view direction of the location is.
- Create a Camera that follows the player so they can explore large maps.

Proposed Solution

Drafting the Map

- Create a layout of the mansion in SweetHome 3D, labeling major and minor rooms.
    - Minor rooms will have 1-2 views each (ex. air ducts, tunnels, walk-in closets, etc).
    - Major rooms will have 4 views each (ex. Bedrooms, kitchen, living room, green house, etc).
- Build every room and its multiple views as Map files, creating sprites from screenshots taken in SweetHome 3D at a consistent angle and height for all views of the room.

Technical

- Create a Map class to hold the map data, the objects it contains, and their positions and layers.
- Store the current room and the previous room in variables in the main file and update as needed.
- An icon displayed in the UI on screen will indicate whether or not the player can rotate the room.
    - In minor rooms, such as hallways, there may not be rotation available so the UI shown on screen must match the end user's expectations for a disabled button (e.g. faded out and semi-transparent, red X over the icon, etc).
- The user clicks the Q or E keys, rotating the room Left (Q) or Right (E), when the rotate icon is enabled.
    - An animation should play when drawing the new map data to make the transition less disorienting.
    - The player character's position should be adjusted to an appropriate X/Y coordinate in the new map to account for the rotation.
- Objects should have consistent states across rooms.
    - e.g. if you pick up an object that's visible in every view of the current location, it will no longer be visible if you rotate the map after picking it up.

Case Files

Coding: Anthony

Testing: Ariel

Overview

We decided as a group to implement support for multiple case files (essentially, quests) and need a way to keep track of the player's progress and what objectives they have or have not completed.

Goals

- Display to the player the current case they're on and a list of objectives for their win condition in a journal section in the menu.
- As the player interacts with the world and discovers clues, update the list of objectives.
- Once all objectives are met, the player can complete a final interaction to win or lose the case.

Proposed Solution

- Like objects and maps,cases (aka "quests") can be stored in a file and deserialized into its own class that will be loaded into memory and manipulated by player input.
- Cases should keep track of a list of objectives that InteractableObjects will be able to manipulate the state of.
- These objectives should have one of three following states:
    - Undiscovered (Displayed in journal as question marks)
    - Discovered (Displayed in journal as an unchecked objective)
    - Solved (Displayed in journal as a checked off objective in the list).
- When all objectives are solved, whatever the win condition is can be completed.
- For murder mystery cases, add support to randomize between a list of Murderers defined in the quest file (matching NPC names).
- Save the current case, current status of objectives, and the randomly chosen killer when the game is saved.

NPCs

Coding: Greyson

Testing: Anthony

Overview

As a group, we wanted to implement other characters for the titular main character to interact with.

Goals

- Implement other characters into the world that the player can talk to.
- Support for branching dialog that can give hints to the player on how to solve the case.

Proposed Solution

- NPCs will be implemented as an extension of UseableObjects
- This would allow for the same features as interactable objects and allow us to utilize existing features:
    - Look
    - Talk
    - Use
        - Using in the case of NPCs could be something like “Accusing of murder” when case objectives are complete
- Dialog will need to be handled in a separate file due to complexity, but prompt/response format from objects can be utilized.

In-Game UI

Coding: Briar

Testing: Greyson

Overview

We need a non-obstructive way to display a lot of information to the player, from dialog to buttons to a player inventory to quest information.

Goals

- Display the following to the player:
    - The current cardinal view and whether the room can be rotated or not.
    - Dialog, interaction strings, and other responses.
    - Character portraits when actively in a conversation to indicate who is speaking.
    - Buttons for interacting with objects and NPCs, and the currently active “object” that the buttons will affect.
    - The current quest and its list of objectives.
    - The player’s full inventory.
    - Which inventory item is the actively selected item if applicable.
- We want to avoid obscuring the scene with the UI as much as possible, as items from the inventory can be used in tandem with items in the world and some items in the world may be small, so we will allow the player to temporarily hide the UI.

Proposed Solution

Design

- By utilizing the detective theming, we can expand upon the placeholder bar from the Cycle One demo, turning it into a stack of “folders” that can show all of this information in a thematically appropriate way while removing the need to design and implement an external series of popups.
- We can have the folders be “pulled out” and expanded to fill the screen, allowing us to display the inventory and the quest text in full without having to have multiple menu windows.

Technical

- The UI class will handle displaying all visible text and prompts to the player.
- Buttons will manipulate the states of InteractableObjects, and InteractableObjects will in turn be able to manipulate the content of what is shown in the text box, what character portrait is shown, etc.

- The camera will keep track of the current active object, the currently selected inventory object, and the player will keep track of an inventory, which can be accessed by the UI to display relevant information.