MVP specification

Architecture

In the context of the Maze Project, the architecture can be simplified as follows:

User Interface (UI): The UI layer is responsible for capturing user input and displaying the game's visuals to the player. It provides controls for camera movement, rotation, and other interactions.

Input Processing: User input from the UI layer is received and processed to determine the intended actions, such as camera movement or rotation. This processed input is then passed to the game logic layer.

Game Logic: The game logic layer handles the core functionality of the maze game. It manages the player's position, camera angle, collision detection, and basic gameplay mechanics. It receives input from the input processing layer and updates the game state accordingly.

Map Data: The map data contains information about the maze layout, including the position of walls and any other relevant game elements. This data can be stored in a suitable format, such as a data structure or file.

Rendering Engine: The rendering engine takes the game state and map data as inputs and generates the visual representation of the game. It uses raycasting techniques to render the walls, textures, and other visual elements based on the player's perspective. The rendered visuals are then sent to the UI layer for display.

Collision Detection: The collision detection component checks for collisions between the player and the walls or other objects in the game world. It ensures that the player cannot move through walls or other solid obstacles. If a collision is detected, appropriate actions, such as sliding along walls, are taken.

Game State Management: The game state management module keeps track of the current state of the game, including player position, level progress, and any other relevant information. It allows for saving and loading game progress and managing transitions between different game states (e.g., menu, gameplay, pause).

Data Modelling

Entities:

Map:

map_id (primary key)
map_name
map_data (stores the layout and structure of the maze)
created_at
updated_at
Player:

player_id (primary key)
player_name
position_x
position_y
angle
created_at
updated_at

Relationships:

One Map can have multiple Players (one-to-many relationship)
This data model represents a basic structure for storing information related to maps and players within the maze game. The Map entity stores details about each map, including its unique identifier, name, data representing the maze layout, and timestamps for creation and updates.

This data model represents a basic structure for storing information related to maps and players within the maze game. The Map entity stores details about each map, including its unique identifier, name, data representing the maze layout, and timestamps for creation and updates.

The Player entity represents individual players within the game. It includes attributes such as the player's unique identifier, name, current position (x, y), angle of view, and timestamps for creation and updates.

The Player entity represents individual players within the game. It includes attributes such as the player's unique identifier, name, current position (x, y), angle of view, and timestamps for creation and updates.

User Stories

Maze Exploration

As a player, I want to navigate through the maze using keyboard controls so that I can explore different paths and reach the exit.

Acceptance Criteria

- The player should be able to move forward, backward, and sideways using the WASD keys.
- The player's movement should be restricted by walls and obstacles within the maze.
- The player should be able to rotate the camera view left and right using the arrow keys.

Wall Collision

As a player, I want to experience realistic wall collision behavior so that I cannot pass through walls and get a sense of the maze's structure.

Acceptance Criteria

- When the player moves towards a wall, they should not be able to pass through it.
- The player's movement should be halted upon collision with a wall.
- If the player tries to move parallel to a wall, they should slide along it instead of passing through it.

Map Customization

As a player, I want to be able to modify the maze map dynamically so that I can create different maze configurations and challenge myself with new layouts.

Acceptance Criteria

- There should be a mechanism to modify the maze map by adding or removing walls.
- The modifications to the maze should be reflected visually in real-time as the player interacts with the controls.
- The modified maze should retain its changes until the game is restarted.

Build your portfolio project (Week 2): MVP Complete

Progress

Progress Rating: 8/10

Explanation of Progress Assessment: This week, I have made significant progress in implementing various features of the Maze Project. I have successfully completed the core functionalities and achieved the planned milestones for this phase of development. The game now includes rendering of walls using raycasting, player movement and rotation, collision handling, and the ability to modify the map. These features are functioning as intended and have been thoroughly tested.

Completed As Planned:

> Window Creation: I have successfully created a window using the SDL2 library to serve as the game interface.
>
> Raycasting and Wall Rendering: I have implemented raycasting techniques to render walls in the game. The walls are drawn with different colors from the ground and ceiling.
>
> Player Movement and Rotation: The player can move forward, backward, and strafe left or right using the keyboard inputs. Additionally, the player can rotate the camera to change the viewing angle.
>
> Collision Handling: I have implemented collision detection to prevent the player from entering walls. The player is now able to slide along the walls instead of stopping abruptly.

Incomplete Aspects:

> Parser: The implementation of the parser to load the map from a file is still pending. Currently, the map is being modified within the code, but I plan to add the functionality to parse the map from an external file.

Textures: The addition of textures to the walls, ground, and ceiling is yet to be implemented. This feature will enhance the visual appeal of the game and provide a more immersive experience for the players.

User Interface (UI): The development of a user interface, including menus, options, and HUD elements, is still in progress. These components will enhance the user experience and provide necessary information during gameplay.

While a significant portion of the planned tasks has been completed, there are still some aspects of the project that require further implementation and refinement. However, the progress made so far is substantial, and the project is on track to meet the planned deliverables.

Challenge

I am working on the Maze Project individually.

Technical Challenge (200+ words):

The most difficult technical challenge I encountered in the second week of the project was implementing the collision handling for the player and walls. While the concept of detecting collisions seemed straightforward initially, the actual implementation posed several complexities.

One of the major challenges was determining the precise intersection point between the player's movement trajectory and the walls in a 2D space. Since the game is based on raycasting, the walls are represented by lines or line segments.

Secondly handling the player's movement along the walls presented challenges in terms of adjusting the player's position and ensuring smooth sliding motion. I had to consider various factors such as the player's velocity, the angle of collision, and the wall's orientation to calculate the correct sliding direction and distance.

Additionally, optimizing the collision detection algorithm to handle large numbers of walls efficiently posed a technical challenge. I had to carefully analyze the performance impact of different approaches and choose an algorithm that balanced accuracy and computational efficiency.

Finally, ensuring seamless collision responses and preventing the player from getting stuck at complex wall configurations added to the technical challenges. Balancing smooth gameplay with accurate collision detection required thorough testing and iterative refinements.

Non-Technical Challenge (200+ words):

The most difficult non-technical challenge I encountered in the second week of the project was managing time effectively and maintaining motivation throughout the development process. As a solo developer, I had to handle multiple aspects of the project, including coding, testing, debugging, and documentation. Balancing these responsibilities while adhering to a timeline was challenging.

One aspect that affected my time management was the learning curve associated with the unfamiliar technologies and libraries used in the project. While I had prior experience with programming, working with specific tools like SDL2 and raycasting required dedicated learning and exploration. Understanding their functionalities and applying them effectively took more time than anticipated.

Another non-technical challenge was maintaining motivation and focus during the development process. As the sole contributor, there were moments when I faced obstacles or encountered errors that seemed difficult to resolve. This led to moments of frustration and a decrease in motivation. Additionally, the absence of immediate feedback from older members in previous cohort made it challenging to stay motivated and engaged.

Despite these non-technical challenges, I was able to adapt and find solutions to keep the project progressing. By effectively managing my time, staying focused, and seeking support when needed, I ensured that the project moved forward despite the difficulties encountered.

To overcome these challenges, I sought guidance from API developers, dedicated time to debugging, and remained committed to the project's success. I leveraged available resources and maintained a proactive mindset.

Author: Adejare Michael
Hire Me!