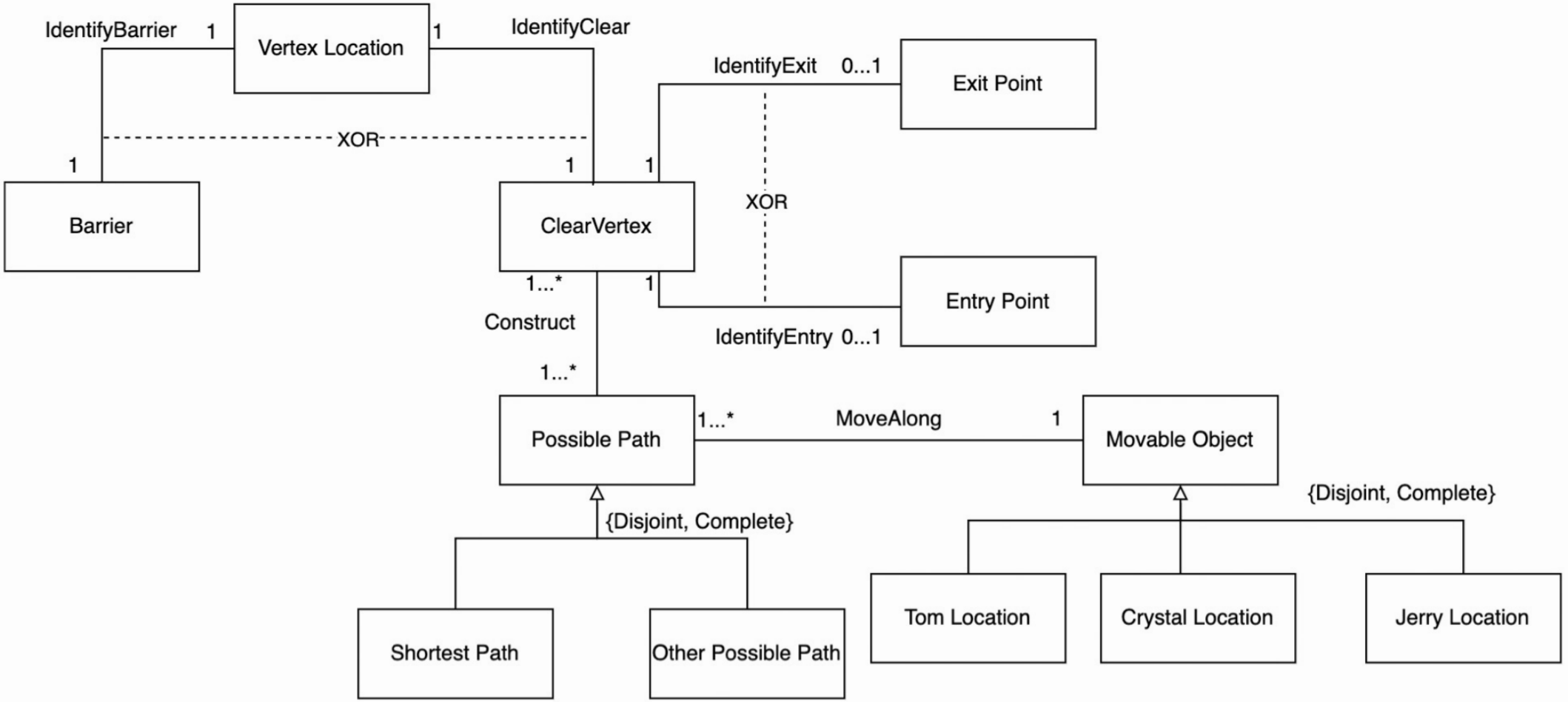
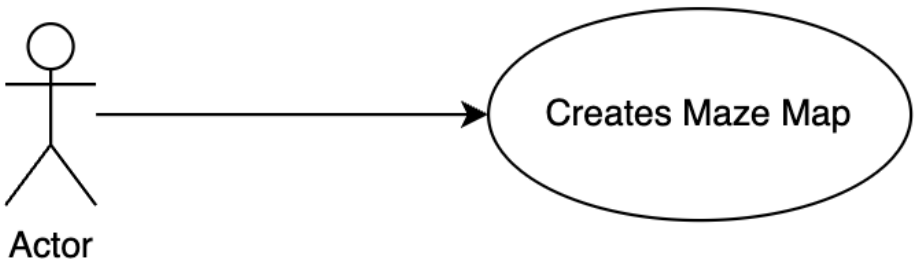
Class Diagram



Remarks: There should be exactly a pair of Entry point and Exit point

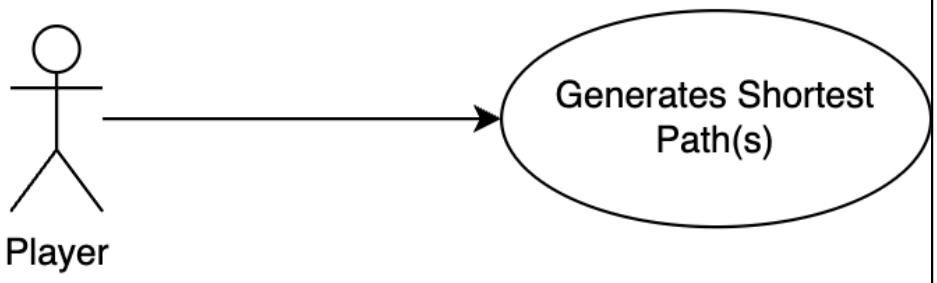
Uses-Case 1: Creates Maze Map

Use-Case Diagram	
Brief Description	This use case describes how the player generates a Maze Map before the game starts
Actor	Player
Precondition	1. The player is on the home page
Basic Flow	1. The use case begins when the player clicks on the button “Create Maze Map” 2. The system displays the interface for the player to select the map-creating method **{Select Map Creating Method}** 3. While the player has a map to create 3.1. If the “Build Map Manually” is selected 3.1.1. Perform subflow **Build Map Manually** 3.2. If the “Import Map” is selected 3.2.1. Perform subflow **Import Map** 4. The system loads and visualizes the Map with white(path) and gray(barrier) grids 5. The system assigns the character Jerry to the entry point and assign Tom to the exit point 6. The use case ends
Subflows	**S1: Build Map Manually** 1. The system initiates a plain 30*30 White PX-Squares **{Begin Building Map Manually}** 2. While the player is editing the map 2.1. If the white cell is clicked 2.1.1. The white cell turns gray 2.2. If the gray cell is clicked 2.2.1. The gray cell turns white **{Confirm Building Map Manually}** 3. The player confirms the map-building

	4. The system converts and saves the manually built map into a CSV file 5. The system informs the player that the manually built map is successfully created **S2: Import Map** **{Begin Importing Map}** 1. The player inputs the CSV file name **{Confirm Importing Map}** 2. The player confirms the import building 3. The system informs the player that the map has been successfully imported
Alternative Flows	**A1: Invalid Map** At **{Confirm Building Map Manually}** or **{Confirm Importing Map}** if Map has less than two valid paths for Jerry to escape or the Map does not have exactly one pair of entry point and exit point shared by all valid paths for Jerry to escape or the or the Map size is not exactly 30*30 1. The system informs the player that the Map is invalid 2. The flow of events is resumed as **{Select Map Creating Method}** **A2: CSV File Not Found** At **{Confirm Importing Map}** if the csv file with does not found 1. The system informs the player that the csv file does not found 2. The flow of events is resumed as **{Begin Importing Map}** **A3: Cancel Map Building** At any point between **{Begin Building Map Manually}** and **{Confirm Building Map Manually}** or between **{Begin Importing Map}** and **{Confirm Importing Map}** 1. The player can cancel the map building 2. The flow of events is resumed at **{Select Map Creating Method}**
Postcondition	1. The Map has at least two valid paths for Jerry to escape 2. The Map has exactly one pair of entry point and exit point shared by all valid paths for Jerry to Escape 3. The Map size is exactly 30*30
Assumption	1. The system can determine whether the map has a valid path and exactly one pair of entry point and exit point

==========

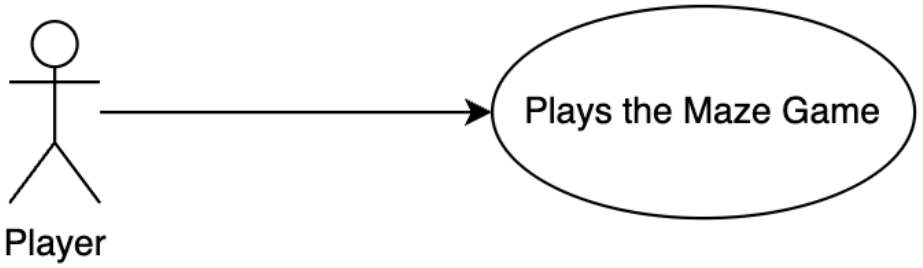
Uses-Case 2: Generates Shortest Path(s)

Use-Case Diagram	
Brief Description	This use case describes how Tom (computer-controlled character) calculates the shortest path between itself and Jerry(User-controlled character) to determine its optimal move to catch Jerry at every instance after the player starts the game
Actor	Player
Precondition	1. The maze has been successfully generated 2. The game has started and not yet ended
Basic Flow	1. The use case begins at regular intervals after the player clicks on the button "Start game" **{Start Generates Shortest Path}** 2. Perform **Explore Possible Path** 3. Perform **Select Shortest Path** 4. The use case ends **{End Generates Shortest Path}**
Subflows	**S1:Explore Possible path** **{Start Explore Possible Path}** 1. The system captures the current position of Tom as the start point and current point 2. The system captures the current position of Jerry as the end point 3. While the current point is different from end point 3.1. If there is any unexplored clear vertex nearby 3.1.1. Change the current point to the location of the clear vertex nearby 3.1.2. The system records the location of current point to a collection of paths 3.1.3. The system marks the current point as "explored" 3.2. If there is no unexplored clear vertex nearby

	3.2.1. Change the current point to the previous location 4. The system discards all the paths that fail to connect the start point and the end point. **{End Explore Possible Path}** **S2: Select Shortest Path** 1. If there is only one possible path 1.1. Select the unique possible path as the shortest path 2. If there is more than one possible path 2.1. The system calculates the number of steps in each possible path 2.2. If there is more than one path having the minimum step 2.2.1. Randomly select one of them as the shortest path
Alternative Flows	A1: **End Game interrupt** At any point between **{Start Generates Shortest Path}** and **{End Generates Shortest Path}** if the game ends 1. The use case ends A2: **Time Limit Exceeds** At any point between **{Start Explore Possible Path}** and **{End Explore Possible Path}** if the time used for generating the shortest path exceeds the buffer time for Tom to update its position 1. The system randomly determines the next move of Tom 2. The use case ends
Postcondition	1. The shortest path cannot cross the barrier
Assumption	1. There is always at least one possible path between Tom and Jerry 2. The position of Tom is updated with a fixed frequency

==========

Uses-Case 3: Plays the Maze Game

Use-Case Diagram	

Brief Description	This use-case describes how the player controls the character Jerry to escape from the chasing of Tom
Actor	Player
Precondition	1. The game has started and not yet ended 2. The maze has been successfully generated
Basic Flow	1. The use case begins when the player clicks on the button "Start game" 2. The game initializes the location of Tom and Jerry **{Start of the game}** 3. While the game has not yet ended 3.1. If the system detects input from the keyboard 3.1.1. Perform subflow **Jerry Movement** 3.2. Perform subflow **Tom Movement** 3.3. Perform subflow **Check End Game Conditions** **{End of the game}** 4. The use case ends
Subflows	**S1: Jerry Movement** **{Start of Jerry Movement}** 1. The system determines the new location of Jerry according to the button pressed by the player 2. If the new location of Jerry coincides to a barrier location 2.1. Jerry does not move 3. If the new location of Jerry coincide to a clear vertex location 3.1. Jerry moves by a block according to the direction provided by the player **{End of Jerry Movement}** **S2: Tom movement** 1. The system generates the Shortest Path 2. The system determines Tom's next move according to the generated shortest path **S3: Check End Game Conditions** 1. Get the current location of Tom, Jerry, and the exit point 2. If Tom's location is the same as Jerry's location 2.1. The System sends out "Game Lose Signal" 3. If Jerry location is the same as exit location 3.1. The System sends out "Game Win Signal"

Alternative Flows	**A1: Pause Game Interrupt** At any point between **{Start of the game}** and **{End of the game}** 1. The player can pause the game 2. The System displays an interface for the player to select further action 3. While the player has not selected a choice 3.1. If the player selects "continue" 3.1.1. The flow of the event is resumed as **{Start of the game}** 3.2. If the player select "end" 3.2.1. Display the text "Thanks for playing" 3.2.2. The use case ends **A2: Invalid keyboard Input** At **{Start of Jerry Movement}**, if the user input is not an arrow key 1. Discard the user input 2. The use case is resumed as **{End of Jerry Movement}** **A3: End Game** At any point between **{Start of the game}** and **{End of the game}**, if the system detects the "Lose Game Signal" or "Win Game Sigal 1. If the Signal is "Lose Game Signal" 1.1. Display the text "You Lose" 2. If the Signal is "Win Game Signal" 2.1. Display the text "You Win" 3. The use case ends
Postcondition	1. Jerry's new location must not coincide with the location of any Barrier
Assumption	1. Tom always follows the shortest path generated from the function