**create\_board():** This function initializes a 10x10 game board as a list of lists, fills it with the character specified in me.map (usually "-"), and randomly places 20 bombs (represented by "1") on the board. It ensures the player's starting position is at (0, 0), and the endpoint is at (9, 9). Using nested list comprehensions (unlike c#), it creates the game board, iterates to place bombs randomly, and sets the player's start and endpoint positions.

**showboard(board):** This function prints the game board row by row, converting the elements to strings and separating each element with a space for a more readable display. It employs a for loop to iterate through each row of the board and within each row, utilizes a list comprehension to convert each element to a string. It then joins the elements with spaces and prints the result.

**player class:** The player class encapsulates the player's characteristics and interactions. It includes attributes like lives, coins, position, control scheme, map, and methods for moving, losing lives, and customizing game elements. This class defines the player's properties and provides methods to interact with the game world. Instances of this class are created to represent the player.

**navigate(self, position, move, board, start=False):** Manages player movement on the game board, checking the validity of the move, updating the player's position, handling collisions with bombs, and detecting when the player reaches the endpoint. It takes the current position, desired move, game board, and an optional 'start' flag. It uses dictionaries to map movement commands to position changes, checks if the move is valid, updates the player's position, and handles collisions and endpoint checks.

**is\_valid\_position(self, position, board):** Checks whether a given position is valid within the game board, ensuring it is within the board's bounds and does not point to an "X," indicating an invalid location.

**add\_item(self, item):** allows the player to add items to their inventory. It takes an 'item' and appends it to the player's inventory list. It appends the given item to the player's inventory list and confirms the addition with a printed message.

**purchase(self, item):** handles item purchases in the shop, checking if the player has enough coins to buy the specified item, deducting the cost, and adding the item to the inventory. It checks the item's cost, deducts it from the player's coins if affordable, and adds the item to the inventory. Displays the result for comformation.

**inventoryView(self):** Displays the player's inventory by iterating through the items and printing each one. It uses a for loop to iterate through the player's inventory and prints each item.

**select\_item(self, item, map=False, character=False, bomb=False):** Enables the player to select items from their inventory and customize game elements like the map, character, or bomb. It checks the 'map', 'character', and 'bomb' flags to determine which game element the item should replace, and assigns the selected item to the appropriate attribute (map, character, or bomb).

**verify(item):** Checks if the given item is a valid option for map designs, characters, or bombs by comparing it to predefined dictionaries. It determines item validity by checking whether it exists in predefined dictionaries of valid options and returns a Boolean.