Karen Liaw

CMIS 1301

Components Description Document

UML Object Diagram Description

The Board class handles all objects and functionality dealing with the game board; essentially, it manages the position of the space objects in the game to easily get their positions when called. Here, an array of GameObjects called boardSpace is declared that will hold the ten board spaces in the game that the player will be moving on. spacePosX, spacePosZ are float variables that will hold the x and z position of each space of the board, and spaceNum is an integer variable that will be used to access the element in the boardSpace at the given value. GetSpacePosX() and GetSpacePosZ() are both methods that return the x and z float value position when called. SetSpaceNumber() method takes an integer value as a parameter and sets the value of the spaceNum variable to the integer value passed to it; this will then change the index to look at the location of a different space GameObject in the array.

The Player class handles all objects and functionality dealing directly with the player object; in general, it gets the player's current position and moves it accordingly. A GameObject called player is declared; this will have the player object in Unity attached to it. The player GameObject will get and change the player's current position. The SetPosition() method takes two float values as parameters and uses them as the new x and z position values for the player; SetPosition() creates a new Vector3 object using the float values passed – it keeps the player object's current y position as this does not need to be changed – and passes that new Vector3 object called newPosition to the MoveTo IEnumerator and starts a Coroutine with MoveTo. The IEnumerator MoveTo() is used to animate the player object moving across the gameboard. This was done to avoid having the player object abruptly changing positions when its x and z values were modified. The MoveTo() takes a Vector3 object as a parameter. It uses a while loop that will – until the player's current position equals that of the Vector3 object passed – continuously move the player's object in the game space at a set speed over the course of time it takes the player to reach the new position. The MoveTo() code was made with reference to https://answers.unity.com/questions/1612420/move-object-along-path-points-after-mouse-click-wi.html which is also noted again in the code's comments. The ResetPosition() method will reset the player's current position to the position specified by the two x and z float values passed to it; this method is different than then SetPosition() method in that it does not implement a Coroutine with MoveTo() and instead instantly changes the player's position without animation when called.

The Dice class handles the functions of the dice for this game, such as rolling the dice to get a random number to determine the amount to move by. It declares two integer variables called diceSides and rolledNumber; diceSides will hold the number of sides that the dice will have and rolledNumber will hold the value of the number rolled from the dice. It also declares a UI Text object called rolledDice; this will be used to display what number was rolled on the dice when the Rol() method is called. The Start()

method initializes diceSides value to three, to meet this game's dice sides requirement; this will be the number of possible values that a player can roll on the dice. The Roll() method gets a random value between 1 and the value of diceSides plus one and assigns it to the rolledNumber variable; this will be the number that the player has 'rolled' when the Roll() method is called. The value of rolledNumber is then assigned to the rolledDice Text object and displayed on screen. GetMoneyLost() and GetSpacesToJump() methods both return the integer value of the rolledNumber variable when called; the number rolled is used to move the forward that many spaces and deducts that much money from the player's current currency. Because of this, GetMoneyLost() returns the negative integer value of the rolledNumber variable to indicate that the value must be deducted from the player's currency.

The Card class deals with creating Card objects that the Deck class will use. The Card class has a string variable called message which will have the action cost and currency cost of that Card when 'drawn' in the Deck class. actionCost and currencyCost are both integer variables that will hold the value for the number of spaces the player will be moving, and the amount of currency that will be deducted or added to the player's current currency amount, respectively. The Card class also has a constructor that takes a string value that will be initialized to the message variable, and two integer values that will be initialized to the actionCost and currencyCost variables, respectively.

The Deck class declares two Card object arrays called playingCards – which will be the master deck that holds all the Cards that can be drawn in the game in order -, and shuffledCards – which will be the deck that holds the Cards from playingCards in a random assortment -, both of which will be ten elements each. The Deck class also declares a UI Text object called cardAction; this will take the message variable from each card and display the string held on the screen for the player. There are also three integers declared called drawnMoneyCost, drawnActionCost, and cardsDrawn. drawnMoneyCost and drawnActionCost will hold the value of the Card object's currencyCost and actionCost, respectively. cardsDrawn will be used to hold the index of shuffledCards array, traverse the array in order, and check when all the shuffledCards elements have been drawn in order to reshuffle as necessary. The Start() method initializes the Card objects in the playingCards array with a message, actionCost value, and currencyCost value. The Shuffle() method uses a for loop to copy the Card object elements from playingCards to the shuffledCards array in random order. The Draw() method uses the cardsDrawn variable to check if the cards must first be shuffled; if cardsDrawn is zero, this means the cards have not been shuffled, or they need to be shuffled, and so the Draw() method will then call the Shuffle() method previously described. The Draw() method will use the cardsDrawn value to access that index's element to get that Card object's currencyCost and ActionCost and assign the values to drawnMoneyCost and drawnActionCost respectively. That same Card's message value is then assigned to cardAction to display on screen for the player to know what actions and currency were taken. Then, cardsDrawn is incremented in order to check the next index's object the next time the Draw() method gets called. The Draw() method also checks the cardsDrawn value after incrementing; when cardsDrawn equals 10, this indicates that the shuffledCards deck has already been completely looked through and cardsDrawn is assigned the value of zero. This way the next time the Draw() method is called and cardsDrawn is checked, it indicates that the Cards must be shuffled again. The ReadMoneyCost() and ReadActionCost()

method simply return the integer values of the drawnMoneyCost and drawnActionCost variables for the Card object that was drawn in the Draw() method.

The AssetManager handles anything dealing with the player's assets, primarily in currency for this game. It declares an integer variable called currency that will hold the player's currency value and a UI Text object called currentCurrency that will be used to display the currency variable value for the player to see. The Start() method initializes the currency variable to ten; this will be the player's starting currency value. The Update() method takes the currency value and assigns it to the currentCurrency Text object as a string to display on screen at all times. The GetCurrency() method, when called, returns an the currency variable's integer value. SetCurrency() takes an integer value as a parameter then adds that value to the currency variable; if the value of currency after adjusting is less than 0, SetCurrency() sets a floor and assigns currency to be equal to 0 because currency cannot be negative. When the ResetCurrency() method is called, the currency value is set back to 10, as it was at the start of the game.

The ScoreManager handles anything dealing with the player's score, which in this game just means the space that the player is currently on on the game board. It declares an integer variable called currentScore that will track what space of the board the player is currently on. The Start() method initializes the currentScore value to one as this is the starting space of the player. SetScore() method takes an integer value as a parameter and adds it to the currentScore value; this indicates the space number that the player object should be on. GetScore() method returns the currentScore's integer value when called. ResetScore() method resets the currentScore value to 1 when called; this is to indicate that the player should be back at space one.

The MovementManager class handles any actions and moves that the player can take throughout the game and handles ending the game and initiating a restart of the game. It declares a Script object for the Board, Deck, Player, Dice, ScoreManager, and AssetManager classes to access and their functions and variables. It also declares four Button objects called diceButton, oneStepButton, drawCardButton, and restarter, which will have the respective Button objects in Unity attached to them; these will be used to check what action the player wants to perform and call the necessary function. It also declares two UI Text objects called actionText and endGameText which will be used to display the actions taken on the screen and let the player know if they have won, respectively. A boolean variable called moved is declared and used to check if the player has made a move or not, which will determine if the player is allowed to draw a card or roll a dice or move one space. It also declares two integer variables called moneyToChange and spaceToChange that will hold the value of money to adjust the player's current currency and current space value by, respectively.

The MovementManager's Start() method handles initializing all the Script objects to get the components of each of their respective classes; it also initializes the Button objects and adds onClick listeners to determine when the player wants to make a move and what action they'd like to take. When the player clicks the diceButton, the ThrowDice() method is called. The oneStepButton calls the TakeAStep() method, the drawCardButton calls the DrawCard() method, and the restarter Button calls

the RestartGame() method. TakeAStep() method first checks if the moved variable is set to false – this will indicate that the player has yet to make a move yet and is allowed to perform the action – and then calls the ImplementAction() function and passes the parameters of 1 and -1 to indicate the player is to move one step and one unit of currency must be deducted. The moved variable value is then set to true to indicate the player has made a move. The ThrowDice() method performs similarly to the TakeAStep() method, but to find the parameters to pass to the ImplementAction() method, the Dice class's Roll() function is called and the randomly rolled number is taken and assigned to the moneyToChange and spaceToChange variables. moneyToChange and spaceToChange are then passed as parameters to the ImplementAction() method where the number of spaces and the money to adjust by will be changed accordingly. When the DrawCard() method is called, it first checks if the moved value is true – this indicates that the player has already moved by either rolling a dice or taking a step – then uses the Deck class's Draw() function to get the first available Card object from the shuffledCards array and then uses the ReadMoneyCost() and ReadActionCost() and assigns the returned values to moneyToChange and spaceToMove respectively. Those values are then passed to the ImplementAction() method to move accordingly.

The MovementManager's ImplementAction() method takes two integer parameters representing the number of spaces to adjust the current space and currency values by. It passes the spaceToMove variable to the ScoreManager class's SetScore() method to adjust the score as needed, then uses its GetScore() method to get the player's current space number and assign to an integer variable called currentScore. It passes the moneyToChange variable to the AssetManager class's SetCurrency() method to adjust the currency as needed. The ImplementActio() method then uses takes the value it got from the ScoreManager class's GetScore() method and passes it to the Board class's SetSpaceNumber method to find the x and z position of that space number and assign those values to two float variables. The x and z position of the desired space object from the Board class is then passed to the Player class's SetPosition() method where the player object's current position is changed accordingly. ImplementAction() also checks the currentScore; if the currentScore (after the previously mentioned changes are made) is equal to 10, it calls the EndGame() method to determine if the player has won the game or not.

The MovementManager's RestartGame() method resets the game back to its original settings when called. It calls the reset methods in the ScoreManager and AssetManager to reset the values of the currency and score to ten and one, and gets the position of the first space object in the Board class and passes that value to the Player class's ResetPosition() method to send the player object back to the first space. The EndGame() method is called only when the player has reached the tenth space on the board, indicated by the ScoreManager's currentScore value. There, the currency value in the AssetManager is checked; if the value is greater than 0, then the AssetManager's SetCurrency() method is called to deduct one more unit of currency from the player's current currency value and the endGameText is changed to announce the player has won. If the player's currency value were to have equaled zero, the player would lose and the endGameText would be changed to display the "You lose" string accordingly.