Maze Experiment Documentation (CAO: 05/7/2020)

By Jacob Augustine, Tyler Dansby and Nick Kaliher

Table of Contents

Overview

Our project is focused on making a maze navigator with audio tasks to measure multitasking performance of solving these cues while navigating the maze. The research assistant will open the program and be able to set a variety of parameters for the experiment. These include maze type (egocentric or allocentric), condition group, number of attempts, time out of maze trial, and number of successful attempts till the next maze. After this, there will be a consent screen for the participant, and then instructions detailing how they will move about each maze and progress through the experiment. The participant will go through the same maze multiple times until either they time out, they run out of attempts, or they have completed it successfully multiple times in a row without error. The next maze is then randomly selected by their condition group. There will also be auditory tasks set in the maze that the participant will have to answer via key press while solving the maze. Once the trials have been completed the user will fill out a short survey. All of the data will be put into an excel sheet - participant ID, data type (D: training, T: testing, S: survey), maze type, maze number, attempt number, errors made in the maze, all movement made, time movement was made, maze location, and survey data - so that the researcher can view them later.

Adding new Mazes and Audio Files

Note: Adding new mazes or audio files will make it so the application needs to be rebuilt. Since there is no database in place, all these changes are local and need to be sent out to participants again.

Mazes

In order to add new mazes, you will need to first bring up the project in Unity. If you are not already in the MainMenuScene, which will be shown in the "Hierarchy" window to the left, you will need to select the "Scenes" folder in the Project window (bottom half of the screen), and then double click on the "MainMenuScene" from there. This will switch to the MainMenuScene. Once there, look in the project Hierarchy (to the left) again, Find the "MainMenu" object which is contained in the MainMenuScene. If you don't see it, you will need to click the triangle next to the MainMenuScene to dropdown all of the elements which it contains. Once you find the MainMenu object, do the same with it and click the dropdown to see what it contains. Do this one more time with "TitleScreen". Then select "MazeCreatorButton" from the list. You should see it appear in the "Inspector" window which contains various things you can edit. Find the name ("MazeCreatorButton") in the inspector, and directly to the left of that there should be a checkbox that is unchecked. Click on this checkbox to activate the Maze Creator button. Once you click it, you should be able to see the button on the Title Screen of the Unity game view. You can now run the Maze Experiment by clicking the play button near the top of the Unity window, and you will have access to the Maze Creator screen. Once you have a maze and have added it to the project, you can also check that it was added by

going to “Resources/Mazes” in the Project window and finding the filename. That is all that is required to add/change mazes.

Audio

In the Maze Experiment/Assets/Resources folder there will be three different folders used for audio. AlloAudio and EgoAudio simply contain all the different audio files all loaded as a .ogg file type. All audio files should be a .ogg and when uploading new ones can be converted to a .ogg using online converters. To add new audio files just move the new file to either AlloAudio or EgoAudio depending on the type and make sure it follows the naming structure of Allo_#.ogg or Ego_#.ogg. All audio files also require an audio answer needed to compare the correct answer with whatever the participant answers during the experiment. These are found in the AudioAnswer folder. AlloAnsweres.txt and EgoAnswers.txt will be used and new answers should be put in order. Each line contains AudioType_# CorrectAnswer. There is a tab between the audio cue number and the correct answer. **DO NOT** use another type of spacing.

Hardcoded Aspects

Maze Creator

Currently there are only two supported maze sizes, 15x15 and 9x9. This can be accessed from the Scenes folder within unity. We chose to keep this outside of the application to prevent users from adding their own mazes. Currently it is set to only contain 15 mazes. Due to the experimental design, most code is reliant on the groups of five mazes.

Possible Dynamic Options: For the maze builder, have a mode for participants and a mode for researchers. Could have password options (role-based authentication) or just different executables. For the 15 max mazes, you could possibly set a dynamic variable to determine how to split up the groups (they are in 3s right now).

Maze Size and Structure

The maze sizes are hardcoded to be 15x15 or 9x9 and the start must be at the top and the finish must be on the bottom. Currently, the code involving coloring, positioning, and rotation of starting position are all dependent on the hardcoded method FindQuadrant in MazeBuilder.cs. There are a bunch of conditional statements that determine whether they are in 1-9 sectors.

Possible Dynamic Options: This could be done dynamically by dividing the two indexes by the number of sectors in each row. You could then determine which ranges to identify sectors for each of the number of sectors you would want. To make this dynamic and

have the start on any side, someone would have to perform some math to simulate a rotation to place them back on top so the sectors can be found from division once again.

Camera Position in Allocentric

The camera for the allocentric is currently placed at a fixed location above the center of a 15x15 maze. This could be dynamically done through code by setting it to the middle by mazeSize and creating some ratio for how high up to place the camera.

Participant Number and Conditions

Currently in MainMenu.cs, after the participant number is read in, it is calculated which parameters will be set based on *ParticipantID % 6.* With this, there are six separate groups for which a number could land in, thus there are six conditionals to check for each. If someone wanted to change the default constraints, they would need to change these conditionals and possibly the modulo as well.

Possible Dynamic Options: This could be done by reading in a CSV that specifies each participant number and the constraints for that number, then setting parameters based on that.

Pseudo-Random Algorithm for Random Placement

The current algorithm for finding a random placement for the test phase of the experiment is pseudo-random. It will find a random placement based on the sectors of the maze (which is also hardcoded) and not allow any of the following positions: the last three sectors (bottom three), the sector where the original start location is, and it can't be

on the same column. For the last exclusion this does not apply if there is a wall in the way.

Number of Attempts for Practice and Testing Phases

Currently, the number of attempts for both the practice maze and the testing phase for mazes are hard coded to 2. These values can be modified by changing the declarations of numPracticeAttempts and numTestingAttempts in the PersistentManager.cs script.

Unity Installation and Setup

1) Install Unity Hub from https://unity3d.com/get-unity/download
2) You will have to first activate a new license in Unity Hub. Log into an account, click on your account in the top right. Go to Manage License -.> Activate New License -> Unity Personal.
3) Go here https://unity3d.com/get-unity/download/archive and be sure to download Unity 2019.3.10 through Unity Hub
    a) Download Microsoft Visual Studio Community if you do not already have it in Dev Tools.

    Windows

    i) Select Universal Windows Platform Build Support and WebGL Build Support under Platforms

    Mac

    ii) Select Mac Build Support and WebGL Build Support under Platforms
4) After the download is complete you will need to add the project folder in the Project tab in Unity Hub. Go to the file location of the Maze Experiment folder and add the whole project. When opening the project be sure to be using the right Unity version if you have more than one.
5) If working on a team, Unity Teams was what we used to make merging and pulling easier. This is not necessary but does make working on the project easier when with multiple members.

Our Unity Structure

Maze Structure

The mazes are read as .txt files and then converted to a NxN size 2D string array. The string consists of two characters. The first character is either a 0 or a 1. The 1 indicates a wall and a 0 indicates no wall. The second character meanings are described below.

I: Intersection, these are points the player can travel to and from.

S: Start, where the player will begin the maze.

F: Finish, where the player must reach to move on.

N: Nothing, no special characteristics.

Example Line of a Maze:

1N 0N 1N 1N 1N 1N 1N 0N 1N 0N 1N 0N 1N 0N 1N

CSV Output Data

Rows of data include:

- ParticipantID
- DataType
    - D: Training phase of maze
    - T: Testing Phase of maze
    - S: Survey data at the end of experiment
- MazeType: Ego or Allo
- MazeNumber: Maze Number from mazeArrayValue
- AttemptNumber
- Movement

    - Ego: Forward, Rotate Right, Rotate Left
    - Allo: Up, Down, Left, Right
- Error
- AudioType: Ego or Allo
- AudioCue: 0 if not playing # of audio if currently playing
- AudioAnswer: Blank when no answer is given y/n otherwise
- Time: mm/dd/yyyy h:mm:ss am/pm
- MazeLocation: Current Location in maze where move was made
- Gender
- VideoGames: Hours playing video games a week

See attached file for example of csv (Full Run Allo.csv)

File Structure

Assets

Materials

Contains all the colors for the maze. The code will change the material of an object to change its color.

Presets

This folder contains all the presets for a unity project. We do not use any presets, but they are quite usual to preserve settings and code over multiple similar objects.

Resources

This folder contains all the files needed to be loaded after a build. This includes the mazes folder, ego/allo audio files folder, audio answers folder. Normal IO operations are not permitted so resources must be used.

Scenes

This folder contains all the scenes for our project. The descriptions for all the scenes are below.

Scripts

This contains all our scripts that are used to run our project. Their descriptions and methods are described below.

Settings

This folder contains preference, account, plugins settings and other unity settings. We did not touch this folder.

TextMeshPro

This folder contains all the default resources that Text Mesh Pro needs to alter text beyond the normal limits of a simple Unity text object.

Packages

This directory contains a bunch of downloads and plugins that came with unity. We have not touched this folder.

Scenes

MainMenuScene

Unity Scene which contains all menu screens up until the first maze is started (TitleScreen. MazeCreatorScreen, ConstraintScreen, ConsentFormScreen, InstructionEgoScreen, InstructionAlloScreen, and InstructionEgoAlloScreen). These are all contained within the MainMenu canvas for holding UI elements. A different screen can be activated manually by using the checkboxes directly to the left of the GameObject name when one of the screens (or GameObjects) is selected. For example, unchecking the box for TitleScreen, then selecting MazeCreatorScreen and checking the checkbox for it will cause the Game/Scene view window to display the MazeCreatorScreen. It also contains an EventSystem which Unity needs to recognize mouse input.

EgocentricMazeScene

Unity scene which contains the maze after it is built with MazeBuilder.cs. This scene is set up for a first-person perspective. The Main Camera is the “player”, or the object that the player controls.

AllocentricMazeScene

Unity scene which contains the maze after it is built with MazeBuilder.cs. This scene is set up for a top-down perspective. The Main Camera is placed above the maze. The Cube is the “player”, or the object that the player controls.

BeginScene

This scene displays a message after the practice mazes have been completed, letting the user know that they are moving on to the experimental trials.

TrialBeginScene

This scene displays a message after a training phase for a maze has been completed, letting the user know that they are moving on to the testing phase.

TrialSecond

This scene displays a message after the first testing run for a maze has been completed, letting the user know that they are moving on to the second testing run.

MazeAttemptScene

This scene displays a message after a maze has been attempted in the training phase, letting the user know that they will be trying the same maze again.

MazeCompletionScene

This scene displays a message after a maze has been completed (after the second testing run), letting the user know that they are moving on to the next maze.

TimeOutScene

This scene displays a message after the specified “time out” time has been reached during the training phase, letting the user know that they are moving on to the testing phase.

BreakScene

This scene displays a message every five mazes, letting the user know that they can take a break.

SurveyScene

Unity Scene which contains all menu screens after the last maze has been completed (SurveyScreen, Debrief, and Thank You). These are all contained within the Survey canvas for holding UI elements. It also contains an EventSystem which Unity needs to recognize mouse input.

Build and Export

To build and export the Unity project, you must first be in Unity on the platform which you wish to build it for. Attempting to build for another platform then compressing and sending everything over has resulted in the application not being recognized as an application. Once you're ready, simply go to File > Build Settings and make sure that the platform is set to "PC, Mac and Linux Standalone" and that the target platform is set to whatever OS you are currently running. On Windows, set architecture to x86_64. You can then press the "build" button at the bottom and select a location where you would like to build it. This may take a few, but once it is done, it should automatically bring up the folder and you can launch the application from there.

Camera & Lighting Settings

Camera Settings

Egocentric

The camera is the “Main Camera” object. It has the default settings. It is set to perspective for the 3D environment. Its position is dynamically moved as the player.

Allocentric

The camera is the “Main Camera” object. It is set to orthographic projection to simulate 2D and we set the size to 10 to allow the whole maze to be in view. The camera is set to be above the center of the maze. This position is not dynamic.

Lighting Settings

Egocentric

For egocentric we have ambient lighting turned on in the Windows > Rendering > Lighting Settings. Without this turned on there are a bunch of pitch-black shadows covering certain walls. This could be prevented by putting a radial display of lights or some variety of this. We used a single directional light above the center of the maze to make the walls look a little more natural and diffused.

Allocentric

For allocentric we turned off ambient lighting and there are no directional lights. The way the player sees the floor is by the blocks lighting up from their material. This is done through their emission color.

Code Documentation

AlloAudio

Description

Used to load in allo audio cues from Resources/AlloAudio. It will take these loaded audio files and play a random one after a random number of moves between 3 and 6. There is then a delay to allow the participant to answer the audio cue with y or n on the keyboard. Once an answer is given it will check if it is correct from Resources/AudioAnswers/AlloAnswers. It will then call InfoTracker to write the answer to the audio cue to the csv file.

Methods

void Start()

Checks if first audio is disabled from the constraints. It then generates the first random number that an audio cue will be played after a number of moves. This method also loads in all of the AlloAudio clips and all of the Allo Audio answers.

void Play(int index)

Plays an audio clip based off of the given random index. The list is not in proper order so to get the actual audio file to compare correct answers it parses the name for the number after the underscore. This is then saved as a global int called audioFileNumber

void Update()

The Update function is called every frame by Unity. First we check if the mazeArrayIndex is between 5 and 9 inclusive as well as seeing if we are

not in the Testing phase of the experiment. Then we need to check if there is an audio cue currently playing so we don't have two playing over one another. If an audio cue isn't playing a counter is used to see how many times a participant moves forward in ego mazes and just moves at all in allo mazes. If the counter equals the random number generated, another random number is used for the size of the Clip array and play is called. A coroutine is used to wait till the audio cue is over. After another random number is generated and the counter is set to zero. If the wait flag is active we need to keep track if the participant answered an audio cue. The answer is compared to what the right answer would be and InfoTracker is called to write the output to the csv file. Total Errors is also added to if the answer is incorrect.

IEnumerator Wait(int time)

Wait uses the inputted allotted time plus a wait delay to keep a flag set called audioWaitActive to keep the counter from running in Update(). It also allows Update() to know that it should be waiting for an answer on the y or n keys. If the participant hasn't answered within the wait delay after the audio cue totalerror is added to. At the end it sets audioWaitActive back to false.

public string GetAudioCheck()

Used to get what audio cue is playing to be written in the csv file. If no audio cue is playing returns "0".

private void GetAudioAnswers()

Reads in correct answers and is called in Start(). These answers are then loaded into PersistentManager.

AllocentricPlayerController

Description

This class deals with all the controls relating specifically to the allocentric (top-down) view. It inherits from the abstract PlayerController class and overrides the update method in that class.

Methods

override void Update()

The Update method is a default method in Unity, so this is called every frame in the AllocentricMazeScene. It essentially checks for key presses, if the player is not already moving. If the player isn't moving, it will call MoveToIntersection and get the next Vector to move to, and if the player is moving, it will simply wait for them to reach the vector to move to before allowing input again.

IEnumerator MovetoIntersection(System.Action<Vector3> callback)

This method is a coroutine because it needs to be able to run across different frames. This method calculates the next intersection to move to based on where the player is and what direction they are trying to move to. The update method then uses the vector returned and tells the player cube object to move there.

Consent

Description

This class is used to make sure the participant reads the consent screen as they must scroll all the way to the bottom. It also is used to figure out what screen to load next. Depending on the constraints the participant will need to see a different instruction screen. These instructions can be for Ego controls, Allo controls, or both.

Methods

public void ChangeToInstructions()

This method is used to decide what instruction screen needs to be loaded next.

void Update()

The Update() method is used to see if the participant scrolls to the bottom of the consent screen and check the box that they agree before they can click the Next button.

Constraints

Description

This class will take in all of the input and selections from the constraint screen GUI. These constraints determine how the experiment will be performed. The methods sanitize the input to ensure it is of the proper data type. The user is only allowed to move forward once all the constraints are filed in.

Methods

void FillParticipantInput(string number)

Will take the participant number input from the GUI as a string and then sanitize the input to ensure it is an int. This will raise an error if it is not an int, preventing the user from moving to the next screen. This is set to a persistent variable.

void FillEgocentricInput(bool ego)

This method will determine whether to include egocentric mazes via a clicked checkbox in the GUI. Can be pressed along with allocentric. This is set to a persistent variable.

void FillAllocentricInput(bool allo)

This method will determine whether to include allocentric mazes via a clicked checkbox in the GUI. Can be pressed along with egocentric. This is set to a persistent variable.

void FillConditionInput(int con)

This method determines which one of the drop-down menu options was selected for the condition. This is set to a persistent variable.

void FillTimeOutInput(string time)

Will take the timeout input from the GUI as a string and then sanitize the input to ensure it is a number. This will raise an error if it is not a number, preventing the user from moving to the next screen. This is set to a persistent variable.

void FillNumAttemptsInput(string attempt)

Will take the number of attempts input from the GUI as a string and then sanitize the input to ensure it is an int. This will raise an error if it is not an int, preventing the user from moving to the next screen. This is set to a persistent variable.

void FillNumSuccessfulAttemptsInput(string attempt)

Will take the number of successful attempts input from the GUI as a string and then sanitize the input to ensure it is an int. Number of successful attempts is about perfect runs through the maze; this is determined by the error algorithm in ErrorChecker. This will raise an error if it is not an int, preventing the user from moving to the next screen. This is set to a persistent variable.

void FillAudioInput(bool audio)

Checks the check box in the GUI to determine whether to enable or disable audio for the experiment. This is set to a persistent variable.

void ClearInput()

This method will reset all of the input boxes and checkboxes once the user switches to or from the screen.

void Update()

This method checks every constraint to ensure they are all filled in. If they are all filled in or selected the “Next” button will appear and allow the user to move to the next screen.

CSVWriter

Description

CSVWriter is used to write the csv for all the data gathered during the experiment. It will create the directory and file if it doesn't already exist. It is then used to write the lines given to it from InfoTracker. It should be noted that Practice mazes will not be recorded in the csv file.

Methods

public bool FirstLineCheck()

This method is used to see if the directory and csv file needs to be created. If the Directory needs to be made it does so in the current path and creates a folder called ExcelResults. This should be in Maze Experiment. It will then write the first line that is all the column headers (ParticipantID, DataType, MazeType, MazeNumber, AttemptNumber, Movement, Error, AudioType, AudioCue, AudioAnswer, Time, MazeLocation, Gender, VideoGame). If the file already exists then it will not write this first line or create the directory/file.

public void Writer(string line)

Used to write a given line from InfoTracker to the csv file.

public string LastLine()

Used in InfoTracker to get the last line of csv and append the needed survey information.

EgoAudio

Description

Used to load in ego audio cues from Resources/EgoAudio. It will take these loaded audio files and play a random one after a random number of moves between 3 and 6. There is then a delay to allow the participant to answer the audio cue with y or n on the keyboard. Once an answer is given it will check if it is correct from Resources/AudioAnswers/EgoAnswers. It will then call InfoTracker to write the answer to the audio cue to the csv file.

Methods

void Start()

Checks if first audio is disabled from the constraints. It then generates the first random number that an audio cue will be played after a number of moves. This method also loads in all of the EgoAudio clips and all of the Ego Audio answers.

void Play(int index)

Plays an audio clip based off of the given random index. The list is not in proper order so to get the actual audio file to compare correct answers it parses the name for the number after the underscore. This is then saved as a global int called audioFileNumber

void Update()

The Update function is called every frame by Unity. First we check if the mazeArrayIndex is between 5 and 9 inclusive as well as seeing if we are not in the Testing phase of the experiment. Then we need to check if there is an audio cue currently playing so we don’t have two playing over one

another. If an audio cue isn't playing a counter is used to see how many times a participant moves forward in ego mazes and just moves at all in allo mazes. If the counter equals the random number generated, another random number is used for the size of the Clip array and play is called. A coroutine is used to wait till the audio cue is over. After another random number is generated and the counter is set to zero. If the wait flag is active we need to keep track if the participant answered an audio cue. The answer is compared to what the right answer would be and InfoTracker is called to write the output to the csv file. Total Errors is also added to if the answer is incorrect.

IEnumerator Wait(int time)

Wait uses the inputted allotted time plus a wait delay to keep a flag set called audioWaitActive to keep the counter from running in Update(). It also allows Update() to know that it should be waiting for an answer on the y or n keys. If the participant hasn't answered within the wait delay after the audio cue totalerror is added to. At the end it sets audioWaitActive back to false.

public string GetAudioCheck()

Used to get what audio cue is playing to be written in the csv file. If no audio cue is playing returns "0".

private void GetAudioAnswers()

Reads in correct answers and is called in Start(). These answers are then loaded into PersistentManager.

EgocentricPlayerController

Description

This class deals with all the controls relating specifically to the egocentric (first-person) view. It inherits from the abstract PlayerController class and overrides the update method in that class.

Methods

override void Update()

The Update method is a default method in Unity, so this is called every frame in the EgocentricMazeScene. Although this class is very similar to AllocentricPlayerController, we felt it needed to be seperate due to a few unique elements of each class. It essentially checks for key presses, if the player is not already moving in any sense (translation or rotation). If the player isn't moving, it will call MoveToIntersection and get the next Vector to move to or call Rotate90. If the player is moving, it will simply wait for them to reach the vector to move to, or the angle they were moving to before allowing input again.

IEnumerator Rotate90(Vector3 byAngles, float inTime)

This method is a coroutine because it needs to be able to run across different frames. It simply rotates the player view until the specified angle is reached.

IEnumerator MovetoIntersection(System.Action<Vector3> callback)

This method is a coroutine because it needs to be able to run across different frames. This method calculates the next intersection to move to based on where the player is and what direction they are trying to move to.

The update method then uses the vector returned and tells the player camera to move there.

ErrorChecker

Description

ErrorChecker is currently used as a visited list of intersections. If a participant ever crosses over an intersection that they already visited that would add to totalErrors in the persistent manager. This count is then reset back to zero when they reach the end of the maze. This is not how Colleen wanted the errors to be accumulated however. The next group would need to do an algorithm similar to Breadth First Search or Depth First Search to get what she wanted. We did not have time for this and to fix this changes would need to be made in the CheckErrors() method of this class to that of BFS or DFS.

Methods

public void CheckErrors()

This method is basically just a visited list. The current location gets stored and is checked if it is in the list, if not it gets added to the visitedList, if it is in the list an error is added to the total.

void Update()

This method is used to update the current location as well as resetting the totalErrors back to zero when the end of the maze is reached.

InfoTracker

Description

This class is used as the main hub of what will be read into the csv file. This class is better described in the methods below. Overall this class is what is used to keep track of a participant timing out, changing maze scenes to place the user back at the beginning of a maze, if a run is considered a perfect run to be added to a counter to reach a constraint, as well as creating the lines that get fed into the csv writer.

Methods

void Start()

Creates a private CSVWriter object and uses it to write the first line of the csv by calling FirstLineCheck().

void Update()

First this method creates a current time called now this is used to see if the user ever times out in the maze. This is used only during the training of the mazes and not during the testing phase. If a user times out a message screen will be displayed to the participant and they will be placed back in the maze. It then will check if the user ever reaches the end of the maze. If so it will see if they made no error during that run and if so perfectRuns will be incremented for this maze. It will then increment currentAttempts and use SceneManager to place them at the start of the maze, randomly in the maze, or in a new maze.

public void SetDirection(string direction, string mazeType, string audioAnswer = "")

This method is used multiple times in four different classes. It is used to create a string that is then written to the csv. It will also not record for the practice mazes and only for experimental mazes. This method is called after each movement is made in both of the player controllers, the direction they moved is set by the controller the maze type they are moving in and audio answer is only used for the Audio classes. Audio type is then created and allocated, audio is set depending on maze index, the first five mazes will be ego audio cues, the next five allo, and the last five will have no audio cues played. The lines are then created below and example output can be seen in the attached csv. When called in the audio classes all the same parameters are used but direction is set to NULL as there is no movement being described here just an audio answer whis is what the last parameter is set to. Again check attached csv for example.

Instructions

Description

This class simply checks the position of the scroll bar for the instructions scene and will set the “Next” button to active to allow the user to move to the next screen.

Methods

void Update()

Called once per frame and will check the position of the scroll bar. If the scroll bar is very close to or at the bottom, it will set the “Next” button to active.

MainMenu

Description

This class checks initial conditions like participant number from the file and sets constraints if necessary. It allows certain participants to set their own constraints. It also creates the list of randomized mazes used for the experiment.

Methods

void QuitGame()

Will exit the application if called.

void ChangeMenuPage()

If a participant number is over 100 the constraint screen will not be shown and the constraints will be determined by modulo 6 (set by file given by colleen). If the participant number is less than 100, they can choose their own constraints.

void InitializePID()

Reads in the participant ID from the given file participantID.txt and sets a persistent variable.

CreateMazeList()

This method randomizes a list of indexes from 1-15 to represent the 15 mazes in our project. It is set to a persistent variable listOfMazes.

MazeBuilder

Description

This large class contains everything for reading the maze file, instantiating the objects of the maze, coloring the sectors of the maze, and random placement/rotation for the test phase.

Methods

bool IsIndexBounded(int mazeSize, int i, int j)

Checks if the given indices are within the bounds 0 to mazeSize.

int FindQuadrant(int i, int j)

Hardcoded conditionals to determine which of 9 sectors the indices. Works for mazeSize 15 and size 9. Returns 0-8 based on which sector the indices returned.

void FindRandomStart()

Finds a random starting location for the testing phase. Avoids the last three sectors, the sector the original starting location was in, and avoids intersections on the same column (so they can't see the exit from the start).

void FindRandomStartDirection()

Gets called by the BuildMaze method. This method finds a rotation for the random start location so that the user does not start looking at a wall.

void GenerateSectorColors()

Creates a randomized list of colors for both ego and allocentric. This list will be used when instantiating the walls/floors and their colors.

void Setup()

This method gets called by PlayerController.cs to call ReadMazeFile, GenereateSectorColros, and BuildMaze. The reason this is called byu another method is to prevent errors from multiple Start functions trying to access sequentially produced data.

string[,] ReadMazeFile()

Reads in the maze file based on the condition and sets the mazeSize and the mazeArray persistent variables. Returns a 2D string array (mazeArray).

void BuildMaze(string[,] mazeArray)

This method is responsible for instantiating all the objects for the maze. It will first build the floor based on maze size. It will then loop through the mazeArray variable and build the walls. It will set variables for starting/end position as well as the start/end text for allocentric. Where the player starts is determined by this method.

void ResetLightColors()

This method will reset all the colors on the floor for the allocentric lighting process.

void ColorAroundIntersection(int i, int j)

Lights a 3x3 area where the intersection is the center, will not go through walls.

void ColorAroundPlayer(int currentX, int currentZ)

This method will color around a player in a 5x5 square with the player in the middle. It will check each direction, if it's an intersection it will call the ColorAroundIntersection method, if it's a wall it will stop, and if neither it will color the floor.

void Update()

Takes current position and calls ColorAroundIntersection for start or at intersection and calls ColorAroundPlayer if not.

MazeCreator

Description

This class is used for the maze creation section of the GUI. It deals with the MazeTile.cs input and writes newly created mazes to the Mazes folder in the Resources directory.

Methods

void FillFileNameInput(string filename)

This method handles the input from the text box, and verifies that what the user typed in is a valid filename. It is dynamically called from the FileNameField game object in the MainMenuScreen.

void Update()

This method simply displays the "Add" button only when a valid filename has been entered.

void AddButton()

This method checks which maze GUI element is active, whether that be a 15 by 15 or a 9 by 9, and then makes a call to FileWriter with that maze array.

void SwapMazeGUI(int value)

This method is called dynamically from the MazeSizeDropdown game object and simply sets a different maze to be displayed depending on what the user selected in the dropdown.

void SetIndexValue(string index)

This method is called from each MazeTile.cs contained within the grid of GUI tiles. It takes a string of three comma-separated integers (e.g. "1,13,0", "0,0,1") which specifies the row, column, and value to set at that position. It essentially modifies the array stored in the MazeCreator class to what the user is seeing on their screen.

void FileWriter(string[,] mazeArray)

This method simply writes whatever maze the user currently sees on their screen to a new file named what the user specified.

void GetIntersections(string[,] mazeArray)

This method is called within FileWriter, just before writing the maze to the file. It finds all the intersections, since the intersections need to be marked with an I, in the maze that the user is trying to upload. This makes it so the user does have to worry about marking the intersections themselves.

MazeTile

Description

This class will determine what to do when a maze tile is clicked on in the maze creator. The class displays the color of the tile and the text upon clicking. It will also update the values for the maze it is building.

Methods

void OnClick()

This method detects a click from a user and sets the multi-toggle tile. It can be clicked up to 3 times to set a hallway, start, or finish. It can be clicked a 4th time to reset. This method will update the GUI with the color and text changes on the tile as well as update the maze variable for building it.

PersistentManager

Description

The PersistentManager is a singleton class which stores the state of the application across all scenes and is one of the most important classes in here. Without it, there would be no real storage besides Unity's PlayerPrefs, thus anything that needs to be tracked through two different scenes must be stored here. It is initialized upon starting the application.

Methods

void Awake()

This function sets all the class variables to default values.

PlayerController

Description

This is the abstract parent class of EgocentricPlayerController and AllocentricPlayerController. It contains the methods that the two classes share and do not need to be changed per class.

Methods

void Start()

This is a built-in Unity method which is called right when the scene it is in is loaded. It first calls the MazeBuilder.cs to set up the maze array to be loaded in, then sets the class variable mazeArray to that. It also sets the starting position for the player.

abstract void Update();

Method to be overridden in child classes.

void CheckFinish()

This method is called when a player reaches the end of their current move, and it resets the movement flags.

bool NearlyEqual(float a, float b, float epsilon)

This method takes two floats and an epsilon, and checks that the two floats are in the range of that epsilon from one another.

SceneManager

Description

This class is attached to every scene and deals with everything relating to scene switching and deciding which is the correct scene to switch to given the current scene and the constraints. All methods in this class are called from either “Next” button presses or conditions being met in InfoTracker.cs.

Methods

void Start()

This method initializes a few class variables to make it easier to work with the variables from PersistentManager.

void GetConstraints()

This method takes a few variables from the PersistentManager, and simplifies it so that a single number can be checked to see whether the constraints are specifying a run of egocentric, allocentric, or both.

void ChangeScene(int scene)

This is the method that does the actual scene changing. It takes in an integer which specifies the index of the scene to switch. It switches to that scene and resets a few flags. These indices are shown in the build settings under “Scenes in Build”.

void SetTimer()

This method sets the initial timer for a new maze. It calculates this by taking the current time and the timeout constraint and adding them together to get the time to finish.

void ChangeFromMainMenu()

This method decides which scene to change to when the main menu scene is the current scene. It is quite simple since this only happens once and can essentially be hard-coded to change to egocentric or allocentric based on constraints. It also sets the timer.

void ChangeFromMazeScene()

This method is the most involved of them all in this class. It decides which scene to switch to after any maze scene (so from Egocentric or Allocentric). First it checks the phase, that being practice, training, or testing, then it checks if any conditions have been reached in the current phase, and if not, it will reset the player to the current maze by going to the MazeAttemptScene.

void ChangeFromInitialBegin()

This method decides which scene to change to from the “Let’s Begin” scene which comes after the practice maze. It resets the timer since it will be a new maze, as well as changing to ego or allo based on the constraints.

void ChangeFromMazeAttempt()

This method simply returns to the previous scene. It also adds the time spent in the transition screen if it is in the training phase, so that the timeout time is still accurate to only being in the mazes.

void ChangeFromTrialBegin()

This method resets the timer, even though it isn't used in training phases, as well as changing back to the previous scene (that being ego or allo). It also resets the current attempts and perfect runs.

void ChangeFromMazeComplete()

This is the other in-depth method of this class. It resets the timer and current conditions, and then it decides the next scene based on whether the current user is doing both ego and allo or not. If they are, it will alternate to the other type and increment the maze index, or if they are on the last maze, it will go to the survey screen.

void ChangeFromTimeOut()

This scene is the only one to cause back to back transition screens (Timeout to Maze Complete). Due to this, it only needs to reset all flags, and change the phase to testing.

SurveyInput

Description

This class takes in all the input from the GUI during the survey screen and records them as persistent variables for later CSV recording.

Methods

void SetMaleToggle(bool value)

If the gender button is clicked for "Male", set persistent gender value to "Male".

void SetFemaleToggle(bool value)

If the gender button is clicked for “Female”, set persistent gender value to “Female”.

void SetOtherToggle(bool value)

If the gender button is clicked for “Other”, set persistent gender value to “Other”.

void SetPreferNotToggle(bool value)

If the gender button is clicked for “Prefer not to say”, set persistent gender value to “Prefer not to say”.

void SetHoursInput(string value)

Takes the input given from the GUI as a string and converts it to an int and sets to a persistent variable.

void Start()

This is a default method in Unity and it is called right when the object it is attached to is generated. In it, we initialize the gender, and get ready to write the last line of the CSV once the user moves on to the next page.

void Update()

This method checks every frame if all the inputs are all filled. If they are, the “Next” button will appear.

public void WriteLine()

This method calls CSVWriter to write the data recorded on the survey screen.