**Notes for future developers**

This document is meant to serve as an informal guide for a quick start to understanding the workflow of the HomeTown Bound game as well as some quirks/bugs to be aware of.

*Written by Michael Probst*

**General Workflow**

There are two main mechanics to the game: (1) player movement and (2) exercises.

1. Player movement: The player is controlled using the *TankController.cs* script (not the most accurate name, but that’s what it was called before I took over). It is set up so that moving the mouse forward moves the player forward and moving the mouse left and right rotates the player. This movement type is intended for a trackball mouse. There are other alternative inputs for movement including WASD, arrow keys, and touch control through on-screen buttons (note: touch controls will only appear if the build target is a mobile device or you are running in the editor).
2. Exercises: These are the 2D puzzles that you do after reaching a floating module number. Though the mechanics of each exercise can be very different, all of them inherit from the base *Exercise.cs* class. Exercises are organized in *Modules.cs* where one exercise is played after the other. There can be as many exercises in a module as you want, and are automatically detected if they are child of the object with the Module component.
   1. Modules: Some modules require extra functionality based on the exercises (typically this includes increasing difficulty by incrementing parameters when an exercise is completed). To do so, all modules inherit from the base *Modules.cs* script similarly to the exercises.
   2. Module Buttons: These should probably be called Exercise Buttons, but here we are. These are the selectable buttons in an exercise. When one is selected, it notifies the current exercise and prompts the exercise to check if it is completed. This component should be added to any UI element that contributes the success of the exercise.
   3. Scoring: Scores are calculated by *ScoreCalculator.cs* which is an instanced entity. Score is determined by how many cognitive impairment (CI) levels off the player is performing. For example if a player performs at a CI level of 4, but is diagnosed at a CI of 2, then their score is 3/5 stars (3=5-(4-2)).
   4. Patient Data: Upon completion of an exercise, the player’s performance is saved in a .csv file using the *SavePatientData.cs* script. At time of writing, this allows players to save up to 3 attempts through the game to see improvement on individual exercises. The first 2 will always be the patients first 2 attempts, but every attempt after that is considered attempt 3.

**Adding new exercises:** If you add a new exercise, you must ensure that a few things are updated:

1. Distracting sounds: these are set up so that distracting sounds are played at a specific exercise ID. If you added a level to the end of the game, you can simply put in the information at the end of the *Assets/ScriptableObjects/DistractingAudioData.asset.* If you insert a new exercise, you may have to do some extra work to determine the ID of your exercise then adjust the audio datas accordingly.
2. CI Data: The CI data serve as a baseline for determining how a patient should perform for each exercise. This includes a *time* in which the exercise should be completed and an *accuracy* goal for the exercise. Similarly to the distracting sounds, you must determine the ID of the new exercise then create an entry for it. Currently, all expectations for all CI levels are based on my performance of the game where the CI level 0 are my exact scores with a bit of a buffer to make it a bit easier. After altering the *Assets/ScriptableObjects/InitialCIData.asset* you **must delete your local data .csv files**. If you don’t, your changes will not be made. Files are located in the application persistent data path *(C:\<your user>\AppData\LocalLow\University of Kentucky\Homebound).* Note that you only need to input the CI requirements for CI level 0 for each exercise, all other CI levels have their criteria automatically generated upon creation of the ci\_data.csv.
3. Update level count in several scripts: it may be obvious when you enter play mode and get an error for out of bounds of an array, but that is because the level count is not correct in the following scripts: *SavePatientData.cs* and *StatisticsManager.cs* (but there may be others I am forgetting).

**Recording gameplay:** Unity recorder is already imported into the project if you need to record footage. There are also CameraOrbit gameobjects which you may utilize to record sweeping shots of the environments.

**Custom artwork:** there is a folder in the svn root folder called custom\_art which includes source files for the artwork I created for this project. The makehuman files will likely not have references to many of the custom objects which may be found on the makehuman community site. If so, you can probably just delete those files, especially if they aren’t useful to you. Other custom characters were created using the O3n system (I think that’s what it’s called) but there is a scene set up for customizing characters (in third party assets folder) and saving them to use later. It’s a little complicated honestly, but hopefully there are enough characters for you to work with.

**Issues to be aware of**

1. Mouse cursor lock:This is something I fought the entire time working on this project. Because the build target is WebGL, the mouse cursor acts strangely when it is not focused on the game window. You may find cases where you see the cursor when it is outside the bounds of the game window, but it is still moving the character around, then when you get to the top of your screen, the character will not be able to move because the cursor has nowhere left to go. I worked around this issue by prompting the user to click within the window when the cursor is unfocused. I may have missed a few use cases, but there might be a better way to fix this (I spent weeks trying to find that better solution and failed).
2. Retrieving patient data: Because the game is a WebGL build, the patient csv files are saved to the browser’s memory (or cache I think?), either way we cannot download the csv to the user’s memory because browsers don’t allow applications to automatically download themselves to computers. There might be a work around for this if you can somehow get the user’s consent to download a file, but there is not a way to do that I know of.
3. Performance: This is a huge issue because WebGL games need to be pretty lightweight to run smoothly. There are still places in the game where there are performance drops, but it is something you will constantly have to be aware of. I found building lighting is not worth it because it makes the game files way too large and would often cause the browser to run out of memory. However, baking occlusion maps is small enough to load and almost essential to getting good runtime performance due to the complex models in LexTown and HomeTown. Code optimization seems to be less necessary, but do make your code as efficient as you can anyway by using the event system rather than probing for conditions in Update loops.

There isn’t a ton in this document, but hopefully this is enough to get started and to point out some of the major things to be aware of. If needed, Dr. Faiola has my contact info if something is too confusing (which it very well may be).

Good luck!