

FLOW FRAMEWORK DOCUMENTATION

Mangkorn Yuan

Table of Contents

1. How to read this documentation.....	pg. 3
2. Terminology.....	pg. 4
3. Scripts.....	pg. 5
a. FlowMain.cs.....	pg. 5
b. FlowCanvas.cs.....	pg. 6
c. FlowWebConfig.cs.....	pg. 10
d. FlowGameConfig.cs.....	pg. 12
e. FlowLineGenerator.cs.....	pg. 14
f. FlowPlayerMovement.cs.....	pg. 15
g. FlowQuestionHandler.cs.....	pg. 16
h. HitLine.cs.....	pg. 18
i. HitLineObject.cs.....	pg. 18
j. QuestionAnswer.cs.....	pg. 19
k. QuestionLine.cs.....	pg. 19
l. Stage.cs.....	pg. 20
m. Target.cs.....	pg. 20
n. TargetPlayer.cs.....	pg. 21
o. DataAlert.jslib.....	pg. 21
4. Web.....	pg. 22
a. Setting Up Guide.....	pg. 22
5. Algorithms.....	pg. 24
a. Simulated Annealing.....	pg. 24
6. Useful/Important Information.....	pg. 26
a. How the Map Editor works.....	pg. 26
b. Store-bought Assets.....	pg. 27
c. Summary of the programming.....	pg. 27

HOW TO READ THIS DOCUMENTATION

- All italicized words in the documentation could be found in the **Terminology** section.
- For each script in the **Scripts** section
 - Description refers to the general purpose of the script.
 - Variable Groupings refer to the listing of variables underneath the Headers (the name of the variable group) in the script.
 - Functions refer to the functions in the script.
 - Custom Classes refer to the custom-made classes for the framework.
 - **Notes:**
 - It is **HIGHLY** recommended that the documentation for the script is viewed alongside the actual script in the project as they both help clarify each other.
 - The documentation for the script **DOES NOT** cover everything in the script (trivial details were omitted from the documentation).
- The **Web** section only offers a general guideline of how the framework was deployed to the web and **DOES NOT** cover the entirety of the web setup process.

TERMINOLOGY

- **FlowMain.cs:** a Unity C# script
- **FlowCanvas.cs:** ...
- **FlowWebConfig.cs:** ...
- **FlowGameConfig.cs:** ...
- **FlowLineGenerator.cs:** ...
- **FlowPlayerMovement.cs:** ...
- **FlowQuestionHandler.cs:** ...
- **Flow:** Refers to the main scripts of the Unity Project.
- **Game Setting(s):** A specific game configuration.
- **Question Setting(s):** A specific question configuration.
- **Game Mode:** A specific player control setting.
- **Line(s):** A Unity object that approaches the player (could consist of *obstacles*, *collectibles*, etc.).
- **Line Unit(s):** A unit in a *Line* (either an *obstacle*, *collectible*, etc.).
- **Patch(es):** A set of *Lines*.
- **Map Editor:** A GUI for editing *Patches*.
- **Obstacle(s):** What the player should avoid.
- **Collectible(s):** What the player should approach.
- **Empty:** A empty space (that has no effect to the player).
- **Health:** Player health points.
- **Score:** Player score.
- **Question Line(s):** A Unity Object that asks the player a question.
- **Question Answer(s):** An answer (Unity object) to a question.
- **Question Queue:** A list of questions waiting to approach the player.
- **Target(s):** The five disks in the Rhythm Mode.
- **Beat(s):** A ball (Unity object) that is “hit” when the player presses the correct *Target*.
- **Hits:** The score of the player (amount of times a *Beat* was hit).
- **Misses:** Amount of times a *Beat* was missed.
- **Accuracy:** *Hits* divided by the total number of *Beats*
- **Stage(s):** The starting point and destination point of *Lines* (implying that the player lies within).
- **Wall:** The boundaries of the player (so that the player doesn’t fly off camera).
- **Loading Screen:** GUI
- **Starting Screen:** ...
- **Password Screen:** ... (password is “coconutpie”)
- **Pause Screen:** ...
- **GameEnd Screen:** ...
- **Pick Settings Screen:** ...
- **Game Config Screen:** ...
- **Questions Config Screen:** ...

SCRIPTS

FlowMain.cs

- Description
 1. Script that drives the entire game.
- Variable Groupings
 1. **DEBUG:** Contains options for debugging the framework.
 2. **Global Settings:** Contains the state of the framework.
 3. **Main Scripts:** Contains references to *Flow* scripts.
 4. **Other Scripts:** Contains references to scripts.
 5. **Audio References:** Contains references to audio clips.
 6. **Other References:** Contains miscellaneous references.
- Functions
 1. **Start()**
 - Handles references and initiates downloading *Game Settings* and *Question Settings* from the web.
 2. **ApplyGameSettings()**
 - Based on a *Game Setting* – the setting is applied to the current game state. In other words, the game state (variables, configurations, etc.) is switched selected *Game Setting*.
 - Parameters
 1. **int i:** The specific *Game Setting* to be applied to the current game.
 3. **StartGame()**
 - Starts the game by
 1. calling **ApplyGameSettings()** from *FlowMain.cs*
 2. calling **Initialize()** functions from other *Flow* scripts
 3. calling **StartGame()** from *FlowLineGenerator.cs*
 - Parameters
 1. **int mode:** The specific *Game Mode* to be played. If the value is -1, game is set to Rhythm mode.

FlowCanvas.cs

- Description
 1. Script that deals with the game canvas/UI.
 2. Makes use of the JavaScript library (to bring up a web browser popup)
- Variable Groupings
 1. **References:** Contains references to other game objects in the scene.
 2. **Settings (Editor Only):** Contains gameplay settings that are to be configured within the Unity Editor.
 3. **Settings (To be configured by Game Settings):** Contains gameplay settings that will be set automatically by a specific *Game Setting*.
 4. **Status (No Reset):** Contains values that do not get reset for the entirety of the application.
 5. **In-Game Classic:** Contains references to the UI game objects and values for the “classic” *Game Mode*.
 6. **In-Game Rhythm:** Contains references to the UI game objects and values for the “rhythm” *Game Mode*.
 7. **Loading Screen:** Contains references to the UI game objects and values for the *Loading Screen*, the panel that shows up when the game is loading.
 8. **Starting Screen:** Contains references to the UI game objects and values for the *Starting Screen*, the panel that shows up at the start of the game.
 9. **Password Screen:** Contains references to the UI game objects and values for the *Password Screen*, the panel that shows up before granting access to modifying game settings.
 10. **Pause Screen:** Contains references to the UI game objects and values for the *Pause Screen*, the panel that shows up when the game is paused.
 11. **GameEnd Screen:** Contains references to the UI game objects and values for the *GameEnd Screen*, the panel that shows up when the game ends.
 12. **Pick Settings Screen:** Contains references to the UI game objects and values for the *Pick Settings Screen*, the panel that navigates further to the *Game Config Screen* and the *Questions Config Screen*.
 13. **Game Config Screen:** Contains references to the UI game objects and values for the *Game Config Screen*, the panel that allows modifying *Game Settings*.
 14. **Questions Config Screen:** Contains references to the UI game objects and values for the *Questions Config Screen*, the panel that allows modifying *Question Settings*.
- Functions
 1. **ReportResults()**
 - Calls the JavaScript library to bring up a web browser popup containing text that could be copied to clipboard.
 - Parameters
 1. **string str:** the text that is to be showed in the web browser popup.
 2. **Start()**
 - Sets the references for the *Line Units* from the *Map Editor*.
 3. **Initialize()**
 - Resets the values for the script.
 4. **GameplayClassicUI_Init()**

- Changes the game UI for the “classic” *Game Mode* based of the settings from the current *Game Settings*.
5. **Update()**
 - Handles pausing the game and zooming in the camera when the player is in movement.
 6. **CanvasButtons()**
 - Handles all the button click events from all the canvases.
 - Parameters
 1. **int button:** The button (belonging to the current active panel) that is pressed.
 7. **BackToMenu()**
 - Sends the player back to the *Starting Screen* and resets the game.
 - Parameters
 1. **prevCanvas:** the current active canvas that is to be hidden (to make way for the *Starting Screen*).
 8. **CanvasGameMode()**
 - Brings up the *Game Settings* Screen. Calls **GameConfig_SettingsToUI()** from *FlowCanvas.cs*.
 - Parameters
 1. **int button:** the specific *Game Setting* to modify.
 9. **CanvasQuestionMode()**
 - Brings up the *Question Settings* Screen. Calls **QuestionsConfig_SettingsToUI()** from *FlowCanvas.cs*.
 - Parameters
 1. **int button:** the specific *Question Setting* to modify.
 10. **CameraZoom()**
 - Zooms in the camera when the player character moves by calling **CamZoom()** from *FlowCanvas.cs*.
 - Parameters
 1. **bool b:** TRUE to zoom in, FALSE to zoom out.
 11. **CamZoom()**
 - Smoothly zooms in/out the camera when the player character moves / stops moving.
 - Note: Function is of type IEnumerator.
 - Parameters
 1. **float fromVal:** current field of view of the camera in degrees.
 2. **float toVal:** the field of view in degrees that the camera should be set to.
 3. **float duration:** time/duration for the camera field of view change from **fromVal** to **toVal**.
 12. **inGameClassicTextChange()**
 - Updates the game UI (for “classic” *Game Mode*).
 - Parameters
 1. **int select:** Selects the specific UI component to modify.
 2. **string text:** Modifies the selected UI component by replacing it with the given string.
 13. **inGameRhythmTextChange()**
 - Updates the game UI (for “rhythm” *Game Mode*).
 - Parameters
 1. **int select:** Selects the specific UI component to modify.

2. **string text:** Modifies the selected UI component by replacing it with the given string.

14. **GameEnd()**

- Brings up the *GameEnd* Screen.
- Parameters
 1. **bool b:** TRUE if the game was won, FALSE if the game was lost.

15. **GameEndShowResults()**

- Calls **ReportResults()** by passing the string returned by **CurrentGameStatus()** from *FlowGameConfig.cs* as a parameter.

16. **GameEndShowResultsDebug()**

- Prints the string returned by **CurrentGameStatus()** from *FlowGameConfig.cs* as a parameter.

17. **GameConfig_SettingsToUI()**

- Sets the *Game Settings* Screen to the current *Game Settings*.
- Parameters
 1. **int i:** The specific *Game Setting*.

18. **GameConfig_UIToSettings()**

- Sets the current *Game Settings* to what is set in the *Game Settings* Screen.
- Parameters
 1. **int i:** The specific *Game Setting*.

19. **GenerateGameConfig()**

- Returns the current *Game Setting* as a formatted string.
- Parameters
 1. **int i:** The specific *Game Setting*.

20. **GameConfig_NextPatch()**

- Switches between *Patches* within the *Map Editor*.

21. **QuestionsConfig_SettingsToUI()**

- Sets the *Question Settings* Screen to the current *Question Settings*.
- Parameters
 1. **int i:** The specific *Question Setting*.

22. **QuestionsConfig_UIToSettings()**

- Sets the current *Question Settings* to what is set in the *Question Settings* Screen.
- Parameters
 1. **int i:** The specific *Question Setting*.

23. **GenerateQuestionConfig()**

- Returns the current *Question Setting* as a formatted string.
- Parameters
 1. **int i:** The specific *Question Setting*.

24. **IntCheck()**

- Returns an int from a string.
- Parameters
 1. **string s:** The string to be converted.
 2. **bool limit:** TRUE to apply int bounds (min/max). FALSE for no bounds (ignore min/max).
 3. **int min:** Minimum value of the int.
 4. **int max:** Maximum value of the int.

25. FloatCheck()

- Returns a float from a string.
- Parameters
 1. **string s:** The string to be converted.
 2. **bool limit:** TRUE to apply int bounds (min/max). FALSE for no bounds (ignore min/max).
 3. **int min:** Minimum value of the float.
 4. **int max:** Maximum value of the float.

26. PatchToString()

- Returns a formatted string from a *Patch*.
- Parameters
 1. **int patch:** The *Patch* to be converted into a string.

27. StringToPatch()

- Sets a specific *Patch* (by setting the value for each *Line Unit* within the *Patch*) based on a string.
- Parameters
 1. **int patch:** The *Patch* to be set.
 2. **string rawString:** The string that is to set the selected *Patch*.

FlowWebConfig.cs

- Description
 1. Script that handles downloading/uploading data from/to the web as well as storing all the *Game Settings* and *Question Settings*.
- Variable Groupings
 1. **Web Addresses:** Contains web addresses and other web names.
 2. **Settings:** Contains general settings.
 3. **Status (To be configured by Gamemode/Questions):** Contains gameplay settings that will be set automatically by a specific *Game Setting* and a specific *Question Setting*.
- Functions
 1. **Initialize()**
 - Calls **InitQuestions()** (from *FlowWebConfig.cs*) and **InitGamemodes()** (from *FlowWebConfig.cs*) as well as initializes the settings panels.
 2. **InitQuestions()**
 - Initializes *Question Settings* by giving default values.
 3. **InitGamemodes()**
 - Initializes *Game Settings* by giving default values.
 4. **ResetQuestions()**
 - Iterates through the *Question Settings* to set the values for each setting.
 5. **GetAllQuestionAndGameConfigs()**
 - Brings up the *Loading Screen* and calls **DownloadAndExtract()** (from *FlowWebConfig.cs*) to download all the *Question Settings* and *Game Settings* from the web.
 6. **ExtractQuestion()**
 - Sets a specific *Question Setting* to what was downloaded from the web.
 - Parameters
 1. **int i:** The specific *Question Setting*.
 7. **ExtractGame()**
 - Sets a specific *Game Setting* to what was downloaded from the web.
 - Parameters
 1. **int i:** The specific *Game Setting*.
 8. **StringToBool()**
 - Returns a bool from a string.
 - Parameters
 1. **string s:** a value of “True” will return TRUE, anything else will return FALSE
 9. **GetQuestionConfig()**
 - Brings up the *Loading Screen* and calls **DownloadAndExtract()** (from *FlowWebConfig.cs*) to download a specific *Question Setting* from the web.
 - Parameters
 1. **int i:** The specific *Question Setting*.
 10. **GetGameConfig()**
 - Brings up the *Loading Screen* and calls **DownloadAndExtract()** (from *FlowWebConfig.cs*) to download a specific *Game Setting* from the web.
 - Parameters

1. **int i:** The specific *Game Setting*.

11. **PostQuestionConfig()**

- Brings up the *Loading Screen* and calls **UploadAndExtract()** (from *FlowWebConfig.cs*) to upload a specific *Question Setting* to the web.
- Parameters
 1. **int i:** The specific *Question Setting*.

12. **PostGameConfig()**

- Brings up the *Loading Screen* and calls **UploadAndExtract()** (from *FlowWebConfig.cs*) to upload a specific *Game Setting* to the web.
- Parameters
 1. **int i:** The specific *Game Setting*.

13. **DownloadAndExtract()**

- Retrieves data from the web and then calls **ExtractQuestion()** (from *FlowWebConfig.cs*) and/or **ExtractGame()** (from *FlowWebConfig.cs*).
- Note: Function is of type *IEnumerator*.
- Parameters
 1. **bool questionDownload:** TRUE to download *Question Setting(s)*.
 2. **bool questionAll:** TRUE to download ALL *Question Settings*.
 3. **int questionIndex:** The specific *Question Setting*. This parameter is ignored if **questionAll** is set to TRUE.
 4. **bool gameDownload:** TRUE to download *Game Setting(s)*.
 5. **bool gameAll:** TRUE to download ALL *Game Settings*.
 6. **int gameIndex:** The specific *Game Setting*. This parameter is ignored if **gameAll** is set to TRUE.

14. **UploadAndExtract()**

- Uploads data to the web and then calls **ExtractQuestion()** (from *FlowWebConfig.cs*) and/or **ExtractGame()** (from *FlowWebConfig.cs*).
- Note: Function is of type *IEnumerator*.
- Parameters
 1. **bool questionUpload:** TRUE to upload the current *Question Setting*.
 2. **int questionIndex:** The specific *Question Setting*.
 3. **bool gameUpload:** TRUE to upload the current *Game Setting*.
 4. **int gameIndex:** The specific *Game Setting*.

- Custom Classes

1. **GameSettings:** represents a specific *Game Setting*.
2. **QuestionSettings:** represents a specific *Question Setting*.

FlowGameConfig.cs

- Description
 1. Script that manages gameplay settings.
- Variable Groupings
 1. **Settings (Editor Only):** Contains gameplay settings that are to be configured within the Unity Editor.
 2. **Settings (To be configured by Gamemode):** Contains gameplay settings that will be set automatically by a specific *Game Setting*.
 3. **Status (Per game session):** Contains values that get reset after each game.
 4. **Status (No Reset):** Contains values that do not get reset for the entirety of the application.
- Functions
 1. **Initialize()**
 - Resets the values for the script.
 2. **TimeConvertToString()**
 - Converts current game time to a formatted string.
 3. **Update()**
 - Keeps count of time by calling **TimeConvertToString()** from *FlowGameConfig.cs*, updates the game UI, and checks the game time win/lose conditions (ends the game when a specified amount of time is reached).
 4. **PlayerHit()**
 - This function is called whenever the player character collides with a game object (e.g. an *Obstacle*). Depending on the game object, game values (e.g. player *Health*) are modified and **CheckHealth()** (from *FlowGameConfig.cs*) or **CheckScore()** (from *FlowGameConfig.cs*) is called.
 - Parameters
 1. **int type:** the type of object that the player character collided into.
 5. **PlayerHitQuestion()**
 - This function is called whenever the player character collides with a *Question Line*, in other words, whenever the player answers a question by colliding into an answer. Depending on the specific *Question Setting* that produced the *Question Line* in-game, game values (e.g. player *Health*) are modified.
 - Parameters
 1. **int addHealth:** the value to add to *Health* (negative values are allowed).
 2. **int addScore:** the value to add to *Score* (negative values are allowed).
 6. **TargetHit()**
 - This function is called when the *Target* hits/misses a *Beat*. Depending on whether the player hit or missed a beat, the *Hits* counter and *Accuracy* is modified.
 - Note: Only for “rhythm” *Game Mode*.
 - Parameters
 1. **int type:** a hit or a miss.
 7. **CheckHealth()**
 - Checks the game win/lose conditions for *Health* and ends the game when the condition is met (a specified amount of *Health* is reached).
 8. **CheckScore()**

- Checks the game win/lose conditions for *Score* and ends the game when the condition is met (a specified amount of *Score* is reached).

9. **CheckRhythmDone()**

- This function is called when all *Beats* has passed (when the amount of *Hits* and *Misses* add up to the total number of *Beats* for the game session). Ends the game session.

10. **EndGame()**

- Calls **EndGameWait()** from *FlowGameConfig.cs*.
- Parameters

1. **bool b**: TRUE if the player won the game, FALSE if the player lost the game.

11. **EndGameWait()**

- Waits for a specified amount of time, then calls **GameEnd()** (from *FlowCanvas.cs*) with parameter **b**.
- Note: Function is of type *IEnumerator*.
- Parameters

1. **bool b**: TRUE if the player won the game, FALSE if the player lost the game.

12. **CurrentGameStatus()**

- Returns the current game status (*Game Mode*, *Health*, *Score*, etc.) as a formatted string, also called at the end of game sessions to show the result of the gameplay.

FlowLineGenerator.cs

- Description
 1. Script that handles generating the game level (generates *Lines*).
- Variable Groupings
 1. **Settings (Editor Only):** Contains gameplay settings that are to be configured within the Unity Editor.
 2. **Settings (To be configured by Gamemode):** Contains gameplay settings that will be set automatically by a specific *Game Setting*.
 3. **Status (Per game session):** Contains values that get reset after each game.
- Functions
 1. **Initialize()**
 - Resets the values for the script.
 2. **StartGame()**
 - Calls **BuildGame()** from *FlowLineGenerator.cs*.
 3. **BuildGame()**
 - Assigns triggers for *Question Lines* based of the *Question Settings*, calls **ExtractLines()** (from *FlowLineGenerator.cs*), and begins to spawn the first *Line*.
 - Note: Function is of type IEnumerator.
 4. **Update()**
 - Spawns *Beats* at the time mark set for each *Beat*.
 - Note: Only for “rhythm” *Game Mode*.
 5. **ExtractLines()**
 - Extract the *Lines* out of the *Patches* to make them ready to be spawned in-game.
 6. **SpawnLine()**
 - Spawns a *Line*.
 7. **ChangeLineSpeed()**
 - Sets the speed of the stars (flying in the direction towards the player character) in the background of the game.
 - Parameters
 1. **float newSpeed:** the speed to set for the stars.
 8. **SpawnLineRhythm()**
 - Initializes the starting conditions for the “rhythm” *Game Mode* and calls **DelayRhythm()** from *FlowLineGenerator.cs*.
 9. **DelayRhythm()**
 - Delays for a set amount of time and then plays the game music.
 - Note: Function is of type IEnumerator.
 10. **ExtractMusic()**
 - Extract the *Lines* to make them ready to be spawned in-game.

FlowPlayerMovement.cs

- Description
 1. Script that controls the player character movement.
- Variable Groupings
 1. **References:** Contains references to other game objects in the scene.
 2. **Settings:** Contains general settings.
 3. **Status (DO NOT MODIFY):** Contains values that should not be modified manually from the Unity Inspector. The values are made visible only for debugging purposes.
- Functions
 1. **Initialize()**
 - Resets the values and references for the script.
 2. **FixedUpdate()**
 - Handles the “smooth” *Game Mode* by enabling player smooth movement and setting boundaries (so that the player character does not fly off the screen).
 3. **ShakeModel()**
 - Shakes the player character temporarily (to produce a “crash” effect).
 4. **Update()**
 - Handles the “rhythm” *Game Mode* by enabling *Targets* to hit *Beats* and handles the “choppy” *Game Mode* by enabling player choppy movement.

FlowQuestionHandler.cs

- Description
 1. Script that manages *Question Lines* and stores all the *Question Settings*.
- Variable Groupings
 1. **Settings (Editor Only):** Contains gameplay settings that are to be configured within the Unity Editor.
 2. **Settings (To be configured by Questions):** Contains gameplay settings that will be set automatically by a specific *Question Setting*.
 3. **Status (Per game session):** Contains values that get reset after each game.
- Functions
 1. **Initialize()**
 - Resets the values for the script.
 2. **BeginSpawningQuestions()**
 - Calls **SpawnQuestionLoop()** from *FlowQuestionHandler.cs* every second.
 3. **SpawnQuestionLoop()**
 - Spawns the *Question Line* waiting in the *Question Queue*.
 4. **AssignQuestionTriggers()**
 - Assigns a trigger for every *Question Line* based on the *Question Settings*.
 5. **TriggerHealth ()**
 - If *Health* reaches a specified value set for a *Question Line*, add the *Question Line* to the *Question Queue*.
 - Parameters
 1. **int value:** the value to compare with the trigger value.
 6. **TriggerScore()**
 - If *Score* reaches a specified value set for a *Question Line*, add the *Question Line* to the *Question Queue*.
 - Parameters
 1. **int value:** the value to compare with the trigger value.
 7. **TriggerLine()**
 - If the amount of Lines spawned reaches a specified value set for a *Question Line*, add the *Question Line* to the *Question Queue*.
 - Parameters
 1. **int value:** the value to compare with the trigger value.
 8. **TriggerTime()**
 - If the amount of time passed reaches a specified value set for a *Question Line*, add the *Question Line* to the *Question Queue*.
 - Parameters
 1. **float value:** the value to compare with the trigger value.
 9. **SpawnQuestion()**
 - Calls **QuestionInProgress()** (from *FlowQuestionHandler.cs*) if **QuestionIsValid()** (from *FlowQuestionHandler.cs*) returns a TRUE.
 - Parameters
 1. **int i:** the specific *Question Setting* (associated with the *Question Line*).
 10. **QuestionIsValid()**

- Checks if a *Question Line* is valid for spawning according to what is configured/set for the *Question Line* according to its corresponding *Question Setting*.
- Parameters
 1. **int i:** the specific *Question Setting* (associated with the *Question Line*).

11. **QuestionInProgress()**

- Spawns a *Question Line*.
- Note: Function is of type IEnumerator.

12. **QuestionAnswered()**

- This function is called when the player answers the *Question Line*. Calls **QuestionEnding()** from *FlowQuestionHandler.cs*.

13. **QuestionEnding()**

- Calls **QuestionStop()** from *FlowQuestionHandler.cs*.
- Note: Function is of type IEnumerator.

14. **QuestionStop()**

- Resumes spawning *Lines*.

HitLine.cs

- Description
 1. Script that attaches to a *Line*.
- Functions
 1. **Start()**
 - Sets references for the script.
 2. **FixedUpdate()**
 - Makes the *Line* move toward the player character (eventually colliding with or passing the player character).
 3. **LineInit()**
 - Initializes the *Line Units* attached to the *Line*.
 4. **SetTheme()**
 - Sets the “look” of the *Line Units* (e.g. “Space Theme”).

HitLineObject.cs

- Description
 1. Script that attaches to a *Line Unit*.
- Functions
 1. **TargetHit()**
 - Calls **Hit()** from *HitLineObject.cs*.
 2. **InTargetRange()**
 - Sends/removes this script (as a reference) to/from a *Target*.
 - Parameters
 1. **Target t:** the *Target* to send reference to.
 2. **bool b:** TRUE to send reference, FALSE to remove reference.
 3. **Hit()**
 - Disables the *Line Unit* once it has been collided by the player character.

QuestionAnswer.cs

- Description
 1. Script that attaches to a *Question Answer*.
- Functions
 1. **Answered()**
 - This function is called when the player character collides with a *Question Answer* (when the player answers a *Question Line*).

QuestionLine.cs

- Description
 1. Script that attaches to a *Question Line*.
- Functions
 1. **Start()**
 - Sets references for the script.
 2. **InitReferences()**
 - Get references from the *Question Answers* attached to the *Question Line*.
 3. **FixedUpdate()**
 - Makes the *Question Line* move toward the player character (eventually colliding with or passing the player character).
 4. **Answered()**
 - Records the player answer and destroys the *Question Line*.
 - Parameters
 1. **string qAnswer:** the player answer.
 5. **QuestionInit()**
 - Initializes the *Question Line* based of its corresponding *Question Setting*.
 - Parameters
 1. **int addHealth:** the value to add to *Health* when the *Question Line* is answered.
 2. **int addScore:** the value to add to *Score* when the *Question Line* is answered.
 3. **bool[] on:** Sets the *Question Answers* attached to the *Question Line* to be active or inactive depending on values of TRUE or FALSE respectively.
 4. **string[] qAnswers:** The answers for each *Question Answers* attached to the *Question Line*.
 5. **string question:** The question on the *Question Line*.
 6. **FlowMain f:** Reference to *FlowMain.cs*.

Stage.cs

- Description
 1. Script that is attached to the *Stages* of the level.
- Functions
 1. **OnTriggerExit()**
 - This function calls when a *Line* or a *Question Line* has exited the “starting” *Stage* or the “ending” *Stage*. Spawns a *Line* (when the previous spawned *Line* has exited the “starting” *Stage*) as well as destroys a *Line* or a *Question Line* (when they exit the “ending” *Stage*).
 - Parameters
 1. **Collider other:** the collider component of the *Line* or a *Question Line* that exited the stage.

Target.cs

- Description
 1. Script that attaches to a *Target*.
- Functions
 1. **TriggerTarget()**
 - Gets the *Target* to hit the *Beat* on top (if there is no *Beat* on top, then nothing happens, however the special effects of “pushing down” a *Target* will still play).
 2. **OnTriggerEnter()**
 - Adds the *Beat* as reference to the *Target*.
 - Parameters
 1. **Collider other:** the collider component of the *Beat* that entered the *Target*.
 3. **OnTriggerExit()**
 - Removes the *Beat* as reference to the *Target*.
 - Parameters
 1. **Collider other:** the collider component of the *Beat* that exited the *Target*.

TargetPlayer.cs

- Description
 1. Script that attaches to the player character.
- Functions
 1. **Update()**
 - Calls **RayCast()** (from *TargetPlayer.cs*) and **Rotate()** (from *TargetPlayer.cs*).
 2. **Rotate()**
 - Tilts the player character left or right depending on the player character movement.
 - Note: Function is of type IEnumerator.
 - Parameters
 1. **float targetDegrees:** The degree to which the player character should tilt toward.
 3. **OnTrigerEnter()**
 - Depending on what has collided with the player character, act accordingly (e.g. if the player character hits a *Wall* then the player character must not proceed any further).
 - Parameters
 1. **Collider other:** the collider component of the game object that touched the player character.
 4. **OnTriggerExit()**
 - Resets the boundaries of the player character movement when the player character has moved away from a *Wall*.
 - Parameters
 1. **Collider other:** the collider component of the game object that moved away from the player character.
 5. **ShowAnswer()**
 - Sets the UI component attached on top of the player character to a display a message.
 - Parameters
 1. **bool show:** N/A
 2. **string answer:** the string to display on top of the player character.
 6. **RayCast()**
 - Shoots a raycast from the player character frontwards, whenever the player character stands in front of a *Question Answer* (and is within range), the answer from the *Question Answer* is sent to the player character to display.

DataAlert.jslib

- Description
 1. The javaScript code that enables using web browser popups. Used for collecting data from gameplay (by copy pasting from the web browser). The file must be put in Assets/Plugins/...

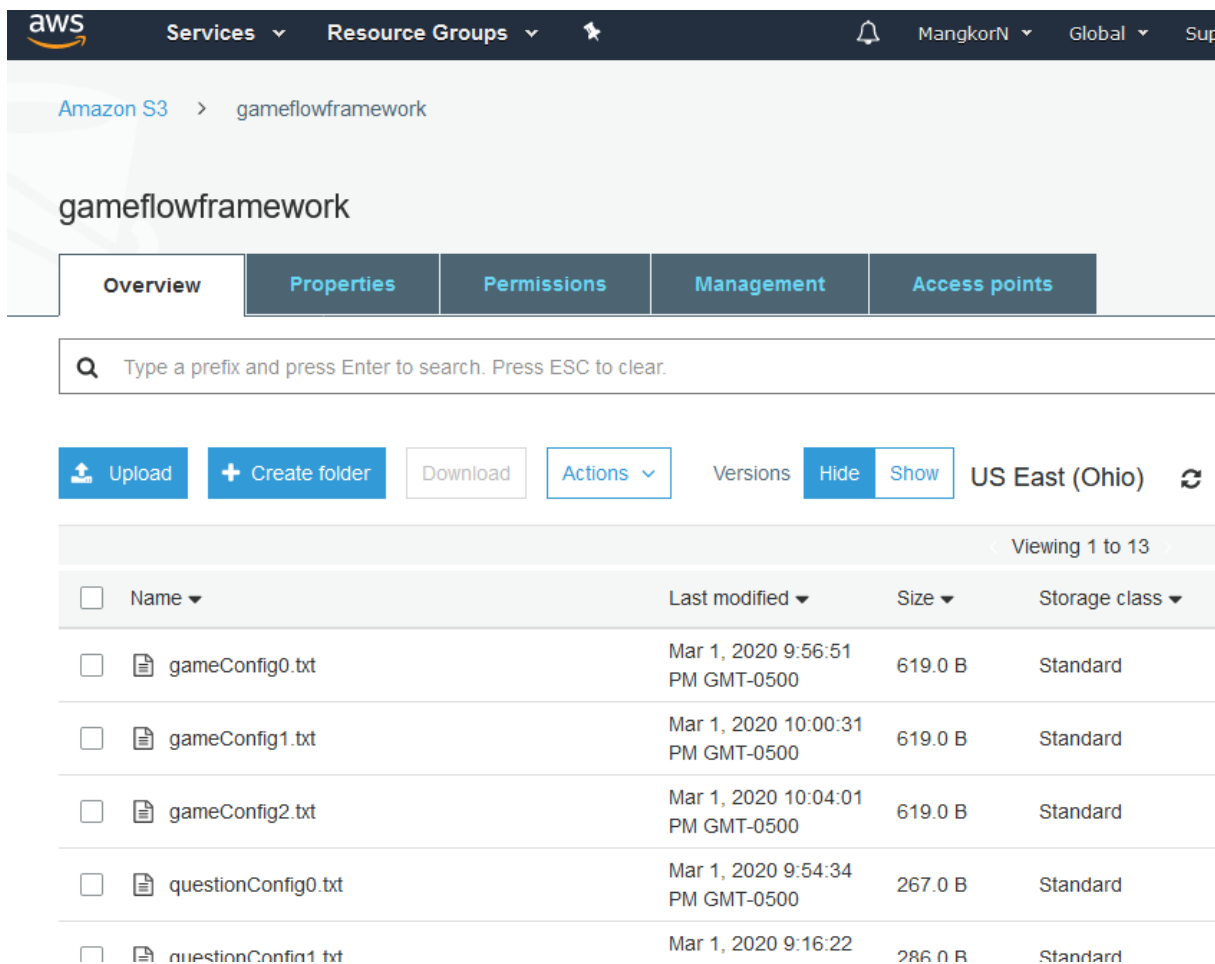
WEB

Setting Up Guide

In summary, for the Flow Project to download/upload *Game Settings* and *Questions Settings*, it communicates with a **website** which in turn communicates with a **database**. The website acts as a middleman that connects the project to the database. The database, as you would expect, contains all the *Game Settings* and *Questions Settings*.

The website files are in the FlowProjectFULL\Web folder, (was) hosted by **Heroku**.

The database is (was) the **Amazon S3**.



The screenshot shows the AWS Management Console interface for an Amazon S3 bucket named 'gameflowframework'. The console displays a list of files with the following details:

Name	Last modified	Size	Storage class
gameConfig0.txt	Mar 1, 2020 9:56:51 PM GMT-0500	619.0 B	Standard
gameConfig1.txt	Mar 1, 2020 10:00:31 PM GMT-0500	619.0 B	Standard
gameConfig2.txt	Mar 1, 2020 10:04:01 PM GMT-0500	619.0 B	Standard
questionConfig0.txt	Mar 1, 2020 9:54:34 PM GMT-0500	267.0 B	Standard
questionConfig1.txt	Mar 1, 2020 9:16:22	286.0 B	Standard

- **Important files for the website**

1. `getGameConfig.php`
 - retrieves *Game Settings* (download from database)
2. `setGameConfig.php`
 - set *Game Settings* (upload/update to database)
3. `getQuestions.php`
 - retrieves *Question Settings* (download from database)
4. `setQuestions.php`
 - set *Question Settings* (upload/update to database)

To change the Flow Project so that the project communicates with another website (perhaps your own private site) or another database (perhaps your own private database) simple modify the PHP files as you see fit.

ALGORITHMS

Simulated Annealing

Warning: NOT ACTIVE IN CURRENT FLOW PROJECT, this algorithm was used in the earlier versions of the Flow Project. It is no longer used, but the file is still in the Unity project for reference.

This algorithm aims to optimize the arrangement of patches in the game according to multiple user defined values/requirements using simulated annealing.

User Inputs:

- **Patch Amount:** the number of patches to be spawned
- **Total Patch Difficulty:** the difficulty that all the patches should try to add up to
- **Iterations:** how many times should we let the algorithm run
- **Easy Patch Difficulty:** how difficult is an “easy” patch
- **Medium Patch Difficulty:** how difficult is a “medium” patch
- **Hard Patch Difficulty:** how difficult is a “hard” patch
- **Extreme Patch Difficulty:** how difficult is an “extreme” patch

Algorithm:

- *difficultyGoal*: the desired difficulty level
- **FOR** $i = 0$ **to** $i < iterations$
 - *currentDifficulty* = total difficulty of all the patches combined
 - *newDifficulty* = new proposed total difficulty as a result of adding a random patch (that is, adding a random patch type into a random index)
 - $t = iterations / (i + 1)$
 - $e_new = | difficultyGoal - newDifficulty |$
 - $e = | difficultyGoal - currentDifficulty |$
 - **IF** ($e_new < e$) **OR** $\text{Exp}(-(e_new - e) / t) \geq \text{Rand}(0,1)$
 - Accept the change
 - **ELSE**
 - Reject the change

Unity C# Code:

```

void GameModeDodger_algorithm3()
{
    //create array of empty patches
    GameModeDodger_initEmptyPatches();

    //keep randomly adding/removing patches until "iterations" is reached
    for (int i = 0; i < iterations; i++)
    {
        int randIndex = Random.Range(0, patchAmount); //random index
        int randType = Random.Range(1, 5); //random type

        //compute current difficulty level
        float currentPatchDifficulty = GameModeDodger_getCurrentDifficulty();

        //compute new proposed difficulty level
        float newPatchDifficulty = currentPatchDifficulty;
        if (patchList[randIndex].type != 0) { newPatchDifficulty -= PatchGetDifficulty(randIndex);

        newPatchDifficulty += GetDifficulty(randType);

        //simulated annealing algorithm components
        float t = iterations / (i + 1);
        float e_new = Mathf.Abs(totalPatchDifficulty - newPatchDifficulty);
        float e = Mathf.Abs(totalPatchDifficulty - currentPatchDifficulty);

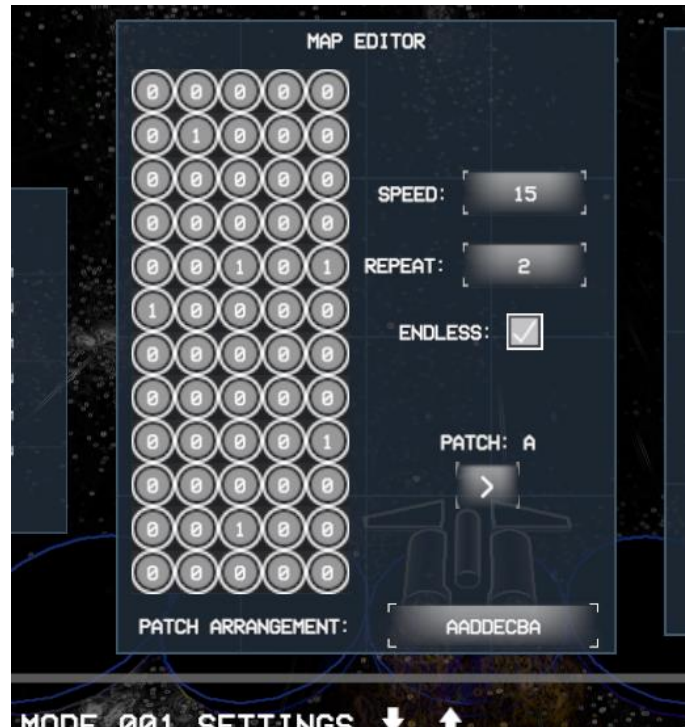
        //accept/reject conditions
        if (e_new < e || Mathf.Exp(-(e_new - e) / t) >= Random.Range(0f, 1f))
        {
            //accept the change
            ReplacePatch(randType, randIndex);
        }
        else
        {
            //reject the change
        }
    }

    //spawn the patches
    GameModeDodger_spawnPatches();
}

```

USEFUL/IMPORTANT INFORMATION

How the Map Editor works



From the *Map Editor* (shown in the picture above), notice **12** rows of *Lines* (each consisting of **5** *Line Units*) totaling up to $12 \times 5 = 60$ *Line Units*. The 12 *Lines* shown in the *Map Editor* represent a single *Patch*. There is a total of **5** *Patches* available (currently), namely, “A”, “B”, “C”, “D”, and “E”. From the *Map Editor*, you may click the arrow to traverse the 5 *Patches* to edit them by manually assigning a **value/number** to each *Line Unit*.

A value/number of **0** indicates *Empty*, **1** indicates *Obstacle*, **2** indicates *Collectible*.

After you are satisfied with the individual *Patch* configurations, the “PATCH ARRANGMENT” dictates the order by which the *Patches* are used (spawned) in-game. E.g.:

Given:

Patch A = 10000...

Patch B = 00001...

Patch arrangement = ABBA

Result (in-game):

10000...

00001...

00001...

10000...

WARNING: Currently there is no protection against game-breaking inputs regarding the *Map Editor*. That is, please make sure that each *Line Unit* has values of **ONLY** 0,1, or 2 - or else the game would break. Additionally, for the *Patch* arrangement, please input only A, B, C, D, or E as characters.

Store-bought Assets

The Unity project contains multiple store-bought assets (some are free to download) from the **Unity Asset Store**. They include:

1. Warp Effect (DIRK JACOBASCH)
2. Modern & Clean GUI (SOTI)
3. Skybox Simple Nebula (Unluck Software)
4. Camera Filter Pack (Vetasoft)
5. Ultimate CRT (Lukasz Lazarecki)
6. Flames of the Phoenix (Onpolyx)
7. Purple Space Nebula Skybox (TL Multimedia)
8. Shuriken Magic Effect Pack (Kalamona)

Summary of the programming

Flow Main drives the entire game by initiating other scripts.

Flow Canvas deals with the game menu and the in-game UI.

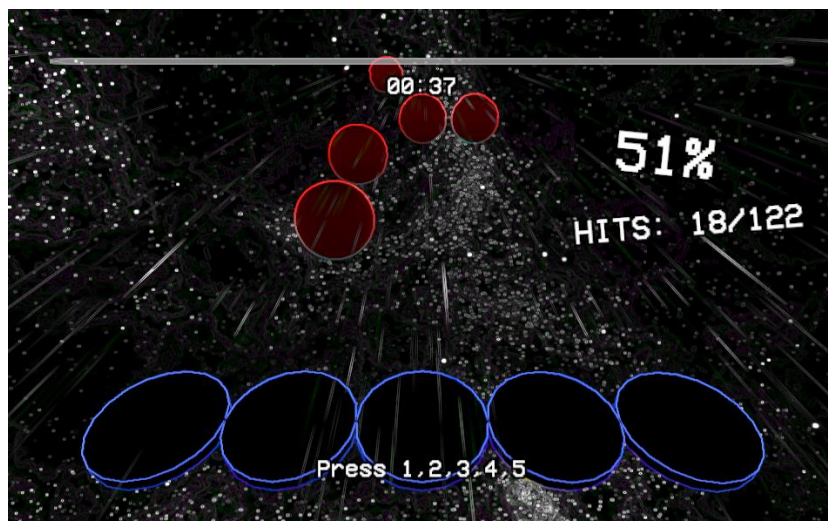
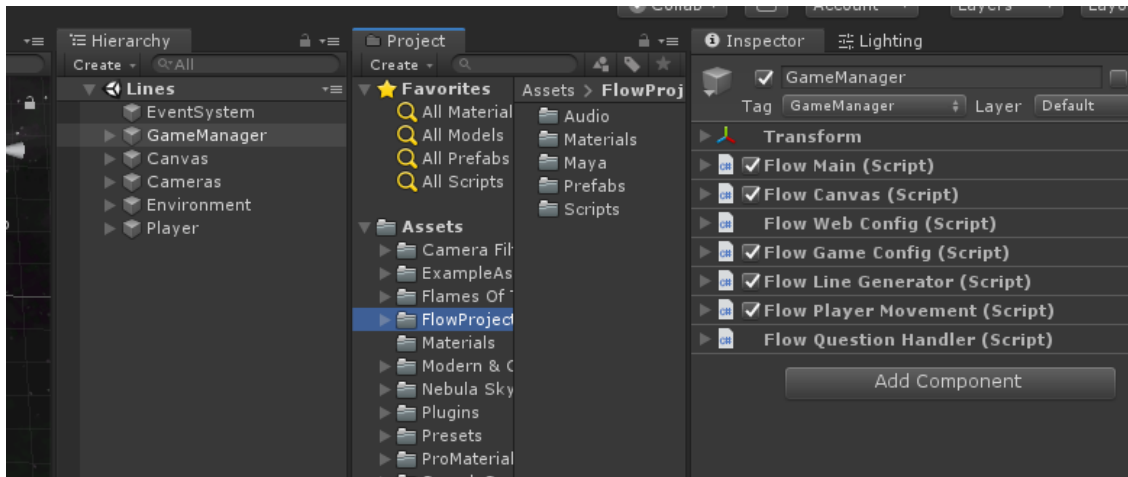
Flow Web Config deals with uploading and downloading from the web.

Flow Game Config deals with gameplay.

Flow Line Generator deals with generating the game level (the *obstacles*, *collectibles*, etc.)

Flow Player Movement deals with player controls/movement.

Flow Question Handler deals with generating *Question Line(s)*.



Contact: mangkorn.yuan98@gmail.com

FLOW FRAMEWORK DOCUMENTATION

Mangkorn Yuan & Garner Newton

TERMINOLOGY

- **FlowMain.cs:** a Unity C# script
- **FlowCanvas.cs:** ...
- **FlowWebConfig.cs:** ...
- **FlowGameConfig.cs:** ...
- **FlowLineGenerator.cs:** ...
- **FlowPlayerMovement.cs:** ...
- **FlowQuestionHandler.cs:** ...
- **Flow:** Refers to the main scripts of the Unity Project.
- **Game Setting(s):** A specific game configuration.
- **Question Setting(s):** A specific question configuration.
- **Game Mode:** A specific player control setting.
- **Line(s):** A Unity object that approaches the player (could consist of *obstacles*, *collectibles*, etc.).
- **Line Unit(s):** A unit in a *Line* (either an *obstacle*, *collectible*, etc.).
- **Patch(es):** A set of *Lines*.
- **Map Editor:** A GUI for editing *Patches*.
- **Obstacle(s):** What the player should avoid.
- **Collectible(s):** What the player should approach.
- **Empty:** A empty space (that has no effect to the player).
- **Health:** Player health points.
- **Score:** Player score.
- **Question Line(s):** A Unity Object that asks the player a question.
- **Question Answer(s):** An answer (Unity object) to a question.
- **Question Queue:** A list of questions waiting to approach the player.
- **Target(s):** The five disks in the Rhythm Mode.
- **Beat(s):** A ball (Unity object) that is “hit” when the player presses the correct *Target*.
- **Hits:** The score of the player (amount of times a *Beat* was hit).
- **Misses:** Amount of times a *Beat* was missed.
- **Accuracy:** *Hits* divided by the total number of *Beats*
- **Stage(s):** The starting point and destination point of *Lines* (implying that the player lies within).
- **Wall:** The boundaries of the player (so that the player doesn’t fly off camera).
- **Loading Screen:** GUI
- **Starting Screen:** ...
- **Password Screen:** (password for standard modes is “coconutpie”) (password for algorithm mode is “AlgorithmConfig”)
- **Pause Screen:** ...
- **GameEnd Screen:** ...
- **Pick Settings Screen:** ...
- **Game Config Screen:** ...
- **Questions Config Screen:** ...

FlowLineGenerator.cs

- Description
 1. Script that handles generating the game level (generates *Lines* for all gamemodes).
- Variable Groupings
 1. **Settings (Editor Only):** Contains gameplay settings that are to be configured within the Unity Editor.
 2. **Settings (To be configured by Gamemode):** Contains gameplay settings that will be set automatically by a specific *Game Setting*.
 3. **Status (Per game session):** Contains values that get reset after each game.
- Functions
 1. **Initialize()**
 - Resets the values for the script.
 2. **StartGame()**
 - Calls **BuildGame()** from *FlowLineGenerator.cs*.
 3. **BuildGame()**
 - Assigns triggers for *Question Lines* based of the *Question Settings*, calls **ExtractLines()** (from *FlowLineGenerator.cs*), and begins to spawn the first *Line*.
 - Note: Function is of type IEnumerator.
 4. **Update()**
 - Spawn *Beats* at the time mark set for each *Beat*.
 - Note: Only for “rhythm” Game Mode.
 5. **ExtractLines()**
 - Extract the *Lines* out of *Patches* to make them ready to be spawned in-game.
 6. **SpawnLine()**
 - Spawns a *Line*
 7. **ChangeLineSpeed(float newSpeed)**
 - Sets the speed of the lines and stars (flying in the direction of the player character) in the background of the game
 - Parameters
 1. **Float newSpeed:** the speed to set for the stars.
 8. **SpawnLineAlgorithm()**
 - Spawns a Line for the Algorithm Mode based on conditions.
 - Algorithm Variables found in *AlgorithmValues.cs*
 9. **ChangeLineSpeedAlgorithm()**
 - Sets the speed of the lines and stars (flying in the direction of the player character) in the background of the game
 - Only works for the Algorithm Mode.
 - Parameters
 1. **Float newSpeed:** the speed to set for the stars.
 10. **AdaptiveDifficultyChange()**
 - Changes the difficulty for the Algorithm Mode if adaptive difficulty is on.
 - Parameters

1. **Int type:** type of adaptive change. (1 = hit rock, 2 = hit star)

11. **SpawnLineRhythm()**

- Initializes the starting conditions for the “rhythm” *Game Mode* and calls **DelayRhythm()** from *FlowLineGenerator.cs*.

12. **DelayRhythm()**

- Delays for a set amount of time and then plays the game music.
- Note: Function is of type IEnumerator.

13. **ExtractMusic()**

- Extract the *Lines* to make them ready to be spawned in-game.