Developer Guide

1. Tools Required:
   1. Unity 2019.3.x or later
      1. Android and iOS Unity Development apk
   2. Python 2.7 or later
   3. IronPython2.7 or later
2. Helpful Tools:
   1. Android Studio to debug Android apks
   2. MacOSX to debug iOS apks
3. Current Issues/Things To Be Aware Of:
   1. When building on Windows, the entirety of the Assets/Plugins/Lib folder needs to be copied into the GameData/Plugins folder where the game was built in order to run the python scripts. Everything in the Assets/Resources folders also needs to be put into GameData/Resources as well.
   2. With the current code, the filepathing cannot be resolved in the Android apk. The internal file system of the apk is different from Windows’ and has been an ongoing unresolved issue. In order for the player to play against the AI in the mobile versions, you will need to find a way to dig into the apk file system and find the location of the Assets/Plugins/Lib to run the python and the python scripts within Assets/Resources to be able to load the AI.
4. Code Design
   1. There are currently 5 C# scripts that the code runs on: Connect4.cs, GameController.cs, QuitToMenu.cs, SceneSwitch.cs, and TicTacToe.cs.
      1. Connect4.cs: This holds all of the internal logic to run the Connect 4 game. This is a singleton class, meaning there will be only one instance throughout every scene.
         1. The script calls Awake() when it is initialized, it will set every button that exists within the scene so they can be shown and hidden throughout the gameplay.
         2. ShowButtons() is contextual and will show appropriate buttons at the league play level depending on who is choosing first.
         3. HideButtons() hides every button and displays the grid.
         4. SetConnect4Board() initializes the game grid explicitly and will typically cause the program to fail if it is not called.
         5. C4Init() is called from the GameController scripts and will start the appropriate game depending on the gamemode.
         6. Connect4Single() will initialize play for a single match of Connect 4, setting the appropriate difficulty for the opponent Agent and creating the board environment.
         7. Connect4Setup() actually starts the matches and is called from Connect4Single() and Connect4League(), setting all of the grid pieces to be blank and randomly assigning who goes first in the match.
         8. Connect4League() sets up everything needed for league play, initializing the league board and setting up a league\_util class to pull the names of the AI from and help generate new AI when a new pairing happens.
         9. NewC4LeagueGame() actually starts a new league match against the AI and will display the name of the paired AI to the screen and set who goes first with regards to league-level decisions.
         10. PlayC4League() plays the current game at the league level, allowing bets to be pleased and showing what the AI chose to do before calling ShowButtons() to continue the game if the AI did not quit.
         11. C4\_AITurn() allows the AI to make a decision from the python script if it is currently the AI’s turn.
         12. ConnectFourSpace() is the listener attached to the grid buttons in the Unity project. When a space is selected by either the player or the AI, a piece will be placed at the lowest possible point in that column and the opposite player will get to go. If the current piece either wins the game, loses the game, or ties the game, appropriate action will be taken and all pieces remaining on the board will be deactivated. After these states have been handled, the game will check the league level (if it exists) and see if either the player or AI has run out of chips. If they have, the game will end and the player can decide to continue or quit. If they haven’t, the next game within the league will begin.
         13. ContinueGame() shows the continue button and allows the player to either play against the next AI opponent or quit back to the main menu.
         14. setChipMul() is a listener attached to the betting buttons and will internally set the multiplier in the python script and externally set the multiplier in the C# script. If the AI chose to quit after this, ContinueGame() will be called. Otherwise, the match will begin.
         15. CallBet() is a listener attached to the Call button and will play the next match without altering the bet multiplier.
      2. GameController.cs: This handles all the main menu functionality and initiates each game depending on which scene is loaded. This script is a singleton, meaning there will be only one instance throughout every scene.
         1. Awake() sets the singleton instance for this class
         2. Start() sets the Easy, Medium, and Hard buttons to hide/display them depending on the currently selected game mode.
         3. OnEnable() sets up the SceneManager in order to dispatch the initialization of each game.
         4. OnSceneLoaded() handles the dispatching of the games. Depending on which scene is being loaded, a different game will start or the main menu buttons will auto select.
         5. getGame() is called from the SceneSwitch script to get the current selected game.
         6. getDifficulty() is called from the SceneSwitch script to get the current selected difficulty.
         7. getGameMode() is called from the SceneSwitch script to get the current selected gamemode.
         8. gameButtonSelect() is a listener and is attached to the game type buttons. It sets the selected button to uninteractable and the unselected button to interactable.
         9. difficultyButtonSelect() is a listener and is attached to the difficulty buttons. It sets the selected button to uninteractable and the unselected button to interactable.
         10. modeButtonSelect() is a listener and is attached to the gamemode buttons. It sets the selected button to uninteractable and the unselected button to interactable.
      3. QuitToMenu.cs: Simple script that returns the user to the main menu
         1. changeScene() called the SceneManager to load the scene given as a parameter, which is the MainMenu scene in use
      4. SceneSwitch.cs: Script which pulls information from the GameController and makes loading decisions based on the selected buttons.
         1. playGame() grabs all of the objects from GameController and loads the selected game’s corresponding scene. It will check if a game is selected and if a difficulty is selected if the game mode is single match. Before loading the scene, it will display the loading screen and attempt to load asynchronously. This typically doesn’t work well and the loading bar progress isn’t updated.
         2. Update() will attempt to progress the loading bar while the game loads but usually doesn’t get the chance to.
      5. TicTacToes.cs: This holds all of the internal logic to run the TicTacToe game. This is a singleton class, meaning there will be only one instance throughout every scene.
         1. The script calls Awake() when it is initialized, it will set every button that exists within the scene so they can be shown and hidden throughout the gameplay.
         2. ShowButtons() is contextual and will show appropriate buttons at the league play level depending on who is choosing first.
         3. HideButtons() hides every button and displays the grid.
         4. SetTicTacToeBoard() initializes the game grid explicitly and will typically cause the program to fail if it is not called.
         5. TTTInit() is called from the GameController scripts and will start the appropriate game depending on the gamemode.
         6. TicTacToeSingle() will initialize play for a single match of Tic-Tac-Toe, setting the appropriate difficulty for the opponent Agent and creating the board environment.
         7. TicTacToeSetup() actually starts the matches and is called from TicTacToeSingle() and TicTacToeLeague(), setting all of the grid pieces to be blank and randomly assigning who goes first in the match.
         8. TicTacToeLeague() sets up everything needed for league play, initializing the league board and setting up a league\_util class to pull the names of the AI from and help generate new AI when a new pairing happens.
         9. NewTTTLeagueGame() actually starts a new league match against the AI and will display the name of the paired AI to the screen and set who goes first with regards to league-level decisions.
         10. PlayTTTLeague() plays the current game at the league level, allowing bets to be pleased and showing what the AI chose to do before calling ShowButtons() to continue the game if the AI did not quit.
         11. TTT\_AITurn() allows the AI to make a decision from the python script if it is currently the AI’s turn.
         12. TicTacToeButton() is the listener attached to the grid buttons in the Unity project. When a space is selected by either the player or the AI, a piece will be placed at the selected position. If the current piece either wins the game, loses the game, or ties the game, appropriate action will be taken and all pieces remaining on the board will be deactivated. After these states have been handled, the game will check the league level (if it exists) and see if either the player or AI has run out of chips. If they have, the game will end and the player can decide to continue or quit. If they haven’t, the next game within the league will begin.
         13. ContinueGame() shows the continue button and allows the player to either play against the next AI opponent or quit back to the main menu.
         14. setChipMul() is a listener attached to the betting buttons and will internally set the multiplier in the python script and externally set the multiplier in the C# script. If the AI chose to quit after this, ContinueGame() will be called. Otherwise, the match will begin.
         15. CallBet() is a listener attached to the Call button and will play the next match without altering the bet multiplier.
5. Left to complete
   1. Dots and boxes
      1. For completion of dots and boxes you will need to implement a command line version of the game in order to help you clearly visualize the game and test the various difficulty settings before running it in unity. The base code for the AI is written in JupyterNotebook form and should be fairly simple to translate to a single python script. There are examples of these python scripts in the Assets/Resources folders.
      2. You will then need to construct a completely functional unity scene in which to play dots and boxes complete with visual assets
      3. Then you will need to implement the AI Q-tables for all three difficulties and verify their functionality
      4. With the single play mode complete you will have to get league play working for dots and boxes including the various betting options and ai opponents
   2. File pathing for Mobile
      1. Currently the games don’t work on mobile. It will build and run, but it can’t find the python scripts or the Q-tables making it completely non-functional
   3. Polish
      1. If you have time after completing the above tasks it would be beneficial to make the games look more impressive be that through the addition of higher quality assets, improved UI, or sound effects.