TECHNICAL SUMMARY OF BEANS CODE

Beans is written in Swift 2.2 programming language. It uses the Objective-C runtime which allows C++ code to run.

Five frameworks are imported. UIKit provides the view architecture needed to manage the application's user interface. QuartzCore provides image processing. SceneKit provides the x,y,z coordinate system and tools for asset manipulation. MultipeerConnectivity employs discovery services of nearby devices over peer-to-peer, wi-fi, and Bluetooth networks by sending message-based data. AVFoundation is the framework used to provide audio management.

Prior to the View Controller class declaration, screenwidth and screenheight of the device is determined and 3 new data types are created--BoardLoc, MoveList, PieceType. BoardLoc is a data type with two intergers, an "x" and a "y". MoveList is a data type that contains an array and the number of items within the array. Each array item holds 2 BoardLocs-fromLoc.x and .y and a toLoc.x and .y. An enum type named, PieceType is created which is a string value. PieceTypes are used to fill the game board array--Empty, Blue, Red, Bean, BlueBean, RedBean.

The class, ViewController, is declared with 2 properties allowing for 2 types of user interfaces. One interface, the UIViewController consists of the user interface used for the entire game except for one small part. That small part is the iOS interface called MCBrowserViewControllerDelegate interface which is a default iOS interface and is only used when the Beans application is called to discover nearby devices.

Next about 50 variables are declared and set as BOOLs, integers, floats, strings. An appDelegate instance is created for multipeer-connectivity usage. Some contain new data types like the enum and structs that were created before the class declaration. One particularly interesting variable, boardArray, is an array of arrays making it a 2D array and is composed of PieceType items. Another 33 variables are added and consist of mp3 and wav files.

The override function viewDidLoad() starts the sequence of events for the program. 4 functions are called in the viewDidLoad. One handles the appDelegate instance, another calls for a BoardArray (the array of arrays). Another starts the background music. The last fills the user interface with the main menu scene.

The rest of the code in this program are a list of functions that are called when a user triggers an event by tapping on various objects in the user interface.

First in that list is the addAppDelegateMultipeerConnectivityHandler() function which controls the appDelegate instance for multipeer connectivity.

Next are several functions used to set up timers before the ai player moves simply because the ai move calculation is instantaneous. 2 other timers are used before a scene transition from game board scene to win/lose scene so the graphics can complete the final piece movement before scene transition.

The menuScene() function first resets gameplay variables and stops the win/lose scene sound effects. It then imports a .dae file which is a 3D scene created in SketchUp. It places a camera focused on the objects within that .dae file and an omni light is created to illuminate the scene. A tapGesture is created and calls for the handleTap() function when any object in the scene is tapped. The menuScene also sets an unused boardArray. This boardArray in the

menuScene is only implemented because the program is expecting one to be there and will crash if is does not exist. Lastly, in this function certain objects are named that include various animations without user interaction. For example “bean01” is connected to the object named “bean01” in the .dae file. This particular object when tapped changes view from menu scene to the first board scene. bean01 as well as bean02 through bean10 objects are then all made invisible by setting their opacities to zero. When the player taps the “versus ai” button, bean01 fades in and user is now able to access the first board scene. After the user successfully wins the first board scene, bean2 fades in and now the user can access the board2Scene, etc.

Next is the teachScene. This scene includes 8 objects of text which explain the rules of the game. These objects occupy the top of the user interface. The bottom half contain a portion of one of the board scenes with game pieces. There are 3 stages of the teach scene. Stage one allows just player movement. Stage 2 allows turn taking between the user and an ai opponent, stage 3, removes turn taking so only the player moves and some more functionality is added because the bean is available to be captured and brought to the home base. Each of the stages create a boardArray and places certain PieceTypes within the 2D array. User can practice piece movement and tap a “next” button to cycle through the text objects explaining gameplay. After the last text placard “next” button is tapped, the menu scene is once again activated.

The next 14 functions are the game board scenes. 10 are used for single player games and 4 for multiplayer. 3 BOOL variables are set at the beginning of these functions and are used to allow or disallow user interaction and to inform the program which gameplay type is employed--single player or multiplayer. Like the menu and teach scenes, a .dae file containing the 3D assets is imported. A second .dae file is imported and named as a subScene. This subScene contains assets used for the submenu. The submenu is a button located in the bottom left hand of the view. For single player, when the user taps the submenu player button two buttons appear allowing the user to restart the current board scene or return to the main menu. For multiplayer boards, the submenu will call a restart and main menu buttons as well as a “how to play multiplayer” overlay and another called “Connect”. The connect button switches the view controller to the one that allows for discovering nearby devices. This view controller has 3 interactive tools. One for searching for devices, one for displaying nearby devices and another for selecting a nearby device. Each of the 14 board scenes contain an individual board array with unique initial piece set up.

The next 4 functions are gameOverYouWin, gameOverYouLose, gameOverPlayerOneWins and gameOverPlayerTwoWins. Each of these scenes contain 2 graphics--the title of the board level completed and a larger image of the bean that existed on the completed board. Tapping the bean will change the bean graphic to one of the bean as a character with legs, arms and face as well as a musical instrument. A quick sound effect of the instrument being played is triggered and then reverts back to the plain bean image. Tapping the title on this view returns player to the main menu scene and the next level becomes available. GameOver scene simply displays the text “You Lose” and a game piece. The piece rotates and contracts in size. Tap “You Lose” text to return user back to Menu Scene with no new levels available. For the multiplayer win or lose scenes, when a game is completed, the device running the program which wins runs the Win scene and the other displays the Lose scene. The win scenes also include 5 particle effects used to create a display of fireworks.

handleTap() is the function called every time the user taps the screen. It first calls an instance of the current scene view and sets a constant called point which can be parsed as floats, point.x and point.y. By turning these floats into integers dividing them by cellHeight and cellWidth respectively the program can determine the BoardLoc to see what space on the board exists at the tap location. This constant is called the tapLoc. Next point.x and point.y are divided by cellWidth and cellHeight again and then multiplied by the size of the game piece to determine locX and locY. These values are used by the program to see if a piece exists at that location and are used to determine where to move pieces to.

Still working in HandleTap function, objects that exist in the various dae files are given a name, usually the same name that is used in the dae file will be used in the code. Next actions are created using SCNActions. These actions allow for scaling the size of objects, moving objects, fading in, fading out, waiting. SCNActions sequences are used which contain a combination of SCNActions.

Next in handle tap, a hit result constant is declared. It is an array populated with all of the objects that are tapped by the user.

The program then checks what object is tapped and then triggers events using the named objects and named actions. For example if user taps “Menu” which is the submenu button that exists on all the board scenes and the teach scene, a sound effect is triggered. “NewGameButton” and “MainMenuButton” are expanded, the recently tapped “Menu” button is shrunk to small to be seen. An “inverse menu” is expanded which is just an invisible object covering the rest of the view is placed. Tapping the “inverse menu” object simply hides itself as well as the 2 recently called “NewGameButton” and “MainMenuButton” and restores the “Menu” button. Many, many more result.node object triggers are listed from adjusting volume to turning on/off piece movement aids to cycling through the teach scene game rules text objects.

Finally we get to the game mechanics which is separated into 2 sections depending on whether multiPlayerRulesOn is true or false. First step is to examine the board array. The user’s tap location (tapLoc) is sent to a function named getTile() which returns the PieceType. If the PieceType is .BluePiece or .BlueBean the selected location (selLoc) is set as the tapLoc. The program then checks which of the user’s blue pieces was tapped and a variable called selectedPiece is changed from a nil value to 1-7 or 10. 1-7 represent normal player pieces and 10 represents the user’s piece holding the bean. If anything other than a player’s pieces is tapped without first tapping one of their blue pieces the handle tap function is ended. If not create a constant called rowCount. rowCount is the returned value of the getRowCount() function. getRowCount() is given the BoardLoc and with a for-loop checks each space in the row of the tapped player’s piece and counts the number of player’s and opponent’s pieces in the same row and returns that value back to the handleTap function. Game pieces can move any direction, orthoganally or diagonally. The distance is determined by the rowCount. Using the rowCount value, the program checks whether the possible destination of the tapped piece is a legitimate move. Moves that would put the piece outside the boundaries of the board and moves that require jumping over other game pieces are all removed. These final possible destination moves call a SCNAction that places a highlight on the legitimate possible destinations. Tapping another of the user’s game pieces stops and restarts this process. When the player then taps one of the legitimate possible destinations, the program checks what exists in that location. It

can be empty, an opponent's piece, an opponent's piece with the bean or the uncaptured bean. The selectedPiece value is used to move the proper game piece and update the boardArray. Landing on an opponent's piece will remove the opponent's piece from the board and update the the number of opponent's pieces. If that value becomes zero, the gameOverYouWin() function is called. If the user moves its piece onto the bean, The bean and the user's piece is removed and the user's piece with the bean is placed in that location. An .Empty PieceType is then placed where the user moved from to update the boardArray. A SCNAction is used to move the object from fromLoc.x, fromLoc.y to toLoc.x, toLoc.y. If the user captures the bean, a SCNAction is used to place a highlight graphic over user's home base. If the user moves the .BlueWithBean piece onto the home base location, gameOverYouWin() is activated. Possible move markers are moved off the board.

After the user has successfully moved a piece moveAI() function is called. Just like handleTap(), moveAI() begins by acknowledging it has control of the most recent scene view and names all of the objects that can be manipulated in this function. If currentPlayer is the human user, "x", currentPlayer is changed to "o". staticPlayer is changed to true so the user cannot move their pieces during this function's activity. Three timers (0.5s, 0.8s, 1.2s) are placed in an array and one is randomly picked. This gives the illusion that the program is thinking. Next the program checks if the bean has been captured, if so who has the bean, how many piece are on the board and how many turns have occurred. From these results the program picks one of 10 preset types of moves. Southern which includes moving a piece straight down or diagonally left-down or right-down, Northern, Eastern, Western. Some move types include directly North and some moves only consider pieces contained within certain rows. Once the program has identified what move type to use, it places all of the refined possible moves into an array. If it has a chance to capture the bean or one of the user's blue pieces, that move is placed as the first item in the array. The program checks if it has a strategic move as the first item, that move is chosen, if not one of the refined moves is chosen at random. The boardArray is updated and the ai game piece object is moved. moveAI() is once again activated but this time it finds that itself, "o", is the current Player so it changes currentPlayer back to "x" and staticPlayer is set to false allowing the user to take their turn.

Multiplayer is just like the single player sequence except at the end of the user's move, moveAI() function is omitted and a dictionary is created consisting of 9 values (selectedPiece, currentPlayer, fromLoc.x and .y, toLoc.x and .y, boardLoc.x and .y, and a string called winnerIs) This dictionary is sent as a JSON object to the connected device that is also running the BEANS application. The values update the connected device's running BEANS application and the process repeats.