LAYPERSON SUMMARY OF BEANS CODE

Beans application is written in Swift programming language. 5 tools are imported to add sound, the rendering and manipulation of game assets and the ability to connect 2 devices together over a wifi or Bluetooth network. Beans is designed to run on iPhones but can also run on iPads and iPad minis.

After the necessary tools, also known as frameworks, are imported there are 3 constants declared. They are used to determine the screen width and height due to the various screen size dimensions of iOS devices the game can be played on. These constants are used to calculate where on the screen a user interaction has taken place.

The next 3 items declared in the code are as follows: (1) BoardLoc (2) PieceType (3) MoveList. BoardLoc is made up of 2 numbers and is used to identify the spaces on the game board. The top left space on the game board is known by the program as 0,0 and the bottom right space on the board would be 6,6 for a game board that is 7 by 7 spaces. It is 7 by 7 and not 6 by 6 because 0 through 6 are 7 values. PieceType is a string value. A string is just a sequence of letters like a word is a sequence of letters. They are limited to be Empty, BluePiece, RedPiece, BlueBean, RedBean, Bean. These strings are types of game pieces that occupy all of the spaces on the game board. MoveList is a specifically designed data type. It is used to contain the list of possible moves the AI, artificial intelligence, may choose from.

The program's class is called GameViewController. It is the name of the file. It can be named BeansGame or GameCode or TimeKiller just like one can name a Word document whatever one chooses. When declaring the class, it must have at least one property which establishes the type of user interface. This program uses 2 types of interfaces. One called UIViewController which is used to load and unload the various views like the MainMenu, the game boards, and the win or lose screens. The other property for the GameViewControlller is called MCBrowserViewControllerDelegate and this is used when a user wishes to connect with another device for multiplayer gameplay. This screen could have also been a UIViewController but the MCBrowserViewControllerDelegate is very useful because it is a predesigned user interface with buttons that allow the user to look for and select other devices.

When the Beans application is opened, the view screen of the device is filled with graphics that makeup the main menu scene--the game title, volume controls, a tutorial button, game level buttons. Tapping the graphics triggers events. For example tapping "Music Volume" button will allow the user to adjust music volume. Tapping the "level one" button will send the main menu screen away and the level one game board will appear on the device's screen. These graphics are imported from a file created in Sketch-Up which is 3D modeling software.

Variables, called vars, are created that are used throughout the application. They are like switches that allow events to occur. One called helpersOn = true allows some code to highlight possible piece move destinations. If the users tap the "Guides" button on main menu screen, they are able to toggle the highlights on or off. 2 other vars are named rows and cols. They are initially set to equal 16 and 9 respectively. Because they are declared as a var and not a let, they are allowed to be changed. Beans contains 3 differently sized game boards. 10 of the vars have the suffix "-Open" like soyOpen, garbanzoOpen or pintoOpen. They are all initially set to false. As a user completes a level, the subsequent level is opened by changing these particular vars to true. The last 20 or so vars in this list are audio files used for background music and sound effects.

After all of the variables are set, the rest of the code is made up of functions. The first one is called the “viewDidLoad()” function and this one is the most significant one. It is used to inform the program what it is supposed to do. It is very short. It is comprised of 5 functions. Here it is:

```
override func viewDidLoad() {
        super.viewDidLoad()
        addAppDelegateMultipeerConnectivityHandeler()
        addBoardArray()
        addBackgrtoundMusic()
        menuScene()
}
```

super.viewDidLoad() doesn’t really do anything but it is best practice to include it. The next function, addAppDelegateMultipeerConnectivityHandeler(), handles the procedure when 2 devices are used for multiplayer gameplay and data is sent from one device to the other. addBoardArray() calls the game board into action. An array is simply a list of elements. It can be a list of numbers or characters or it can even be a list of lists which is the case in the addBoardArray() function. By creating a list of lists, a 2-dimensional game board is created. The first list is the first row of spaces at the top of the game board. The second list expands that one row by creating the columns of the game board. addBackgrtoundMusic() simply starts playing the background music.

menuScene() function loads the elements comprising the main menu screen. It does this by importing a .dae file which is a standard 3-dimensional model file format created in SketchUp. A camera and a light source are also created and placed. A gesture recognizer is called in this function to handle user interaction. It refers to any tap on the screen to the handleTap() function which will be explained later. The menuScene() also stops audio that has not completed from the win and lose scenes. After a user wins or loses a game, users are transitioned to the win/lose screens. From the win/lose scenes, users are transitioned to the main menu scene so that is why it is necessary to stop the sound effects from the win/lose scenes.

The next function is called 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 (PieceTypes). There are 3 stages of the teachScene() function. 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. Users 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 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--singleplayer or multiplayer. Like the menu and teach scenes, a .dae file containing the 3D assets is imported. A second .dae file is also imported with the game board screens 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. This function first calls an instance of the current scene view and sets a constant called "point" which can be parsed as point.x and point.y. By dividing point.x and point.y by cellHeight and cellWidth respectively the program can determine the BoardLoc to see what PieceType exists at that tap location. This constant is called the tapLoc. Next, point.x and point.y are divided by cellWidth and cellHeight 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. Next actions are created called SCNActions. These actions allow for scaling the size of objects, moving objects, fading in, fading out, waiting. SCNAction sequences are used which contain a combination of SCNActions.

Next in handle tap, a hit result constant is declared. Hit-testing is the process of finding objects in a scene located at a specified point. 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 into view, the recently tapped "Menu" button is shrunk too small to be seen.

The handleTap() function also includes the game mechanics for singleplayer and multiplayer. The only difference between them is singleplayer ends with the moveAI() function and multiplayer ends by packaging data regarding the move that was just taken and sending that package to the other connected device running the Beans application in the form of a notification.

How does a game move take place? First the user taps one of their game pieces. Several functions begin. First is rowCount().The rowCount() function uses a for-loop to find out how many other pieces (player's or opponent's) that exist on the same row as the tapped game piece. A for-loop is a block of code that repeats a specified number of times. The number of times the rowCount for-loop repeats depends on the “cols” variable which represents the number of columns on the game board. Game pieces can move in any direction on the gameboard--left, right, up, down, and all 4 diagonal directions. The number of spaces a game piece moves is determined by the value returned by the rowCount() function. The next 8 functions implement the rowCount to examine the 8 directions the piece can move. They use a while-loop which is like a for-loop but stop when a certain condition is met. In Beans, game pieces may not jump over other pieces. So the while-loops quit if a game piece exists between the current moving game piece location and its possible destination location. The while loops also quit when the possible destination location would move the game piece of the board. While-loops that quit during this procedure remove that move possibility from the list of possible moves. Move destination locations that are valid are highlighted unless the user turned off the guides toggle from the main menu changing helperOn from true to false. Tapping another of the user's game pieces before tapping the destination location repeats this process. If the user does tap one of a piece's possible move destination locations, the piece is moved using a moveTo SCNAction. Also the game board 2D array is updated. An Empty value is placed where the piece moved from and the space the piece moved to is changed from Empty or Red or RedBean or Bean to Blue or BlueBean depending on if the game piece has the bean or not.

Every game board has 2 variables associated with the number of game pieces each player has on the board. When a gamePiece lands on one of their opponent's gamePieces, the associated numberOfBluePieces or numberOfRedPieces value is updated. When one of these values becomes zero, the game is either won or lost and transitions to the win/lose scene. When a game piece with the bean lands on its home base, the game level is complete and transitions to the win or lose screen as well.

After the user has successfully moves a game 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”. The staticPlayer variable 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 (list) 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 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.