Developer/Maintenance Documentation

How was this game constructed?

- A checklist of what tasks would need to be completed to implement this game was built up and added to the Product Backlog.
- Developers were delegated tasks from the Product Backlog to complete.
- As the product became more complex, additional tasks were added to the Product Backlog.

Structure of our Product:

- The Product is modularised and follows the Object-Oriented model.
- The code is encapsulated within classes as state and behaviour.
- The Product uses a Client/Server model, with the major processing on the Client-side to reduce throughput and stress on the Server, which runs multiple games.
- The modules were created based on their behaviour and role in the product. Our Final Product contained 11 client-side modules; 'player', 'piece', 'connection', 'board', 'setup', 'constants', 'dice', 'form', 'box and button', 'chat', and our main module, 'client'. There Is only one server file.
- The 'player' module has a Player class, that contains information about the player, including their nickname and colour. Likewise, the 'piece' module contains a Piece class which has information about a certain piece, such as its colour and the position on the board. Player has an attribute called all_pieces which is a list of 16 Piece objects, representing all possible pieces in the game.
- The 'connection' module includes a handler that receives messages from the server and reacts accordingly.
- The 'board' module contains the PyGame code that draws the board and pieces onto the window. It holds a reference to the Player object, and its list of Pieces that it uses to draw them in the correct position on the board.
- There are additional modules such as 'scoreboard', which draws the scoreboard onto the window and 'chat', which creates a chat window that allows players in the game to communicate. Modules such as 'constants' contain values regularly used throughout the other modules.
- The 'client' module creates instances of the appropriate classes, and calls functions from others. Client passes references to instances to functions within other modules, allowing them to work together without having to go through the client.
- The Server is also encapsulated into two classes; Game and Lobby. The Lobby class acts as a lobby and holds a list of Game objects. Any new games created from interaction with a client are appended here.
- Each 'Game' class contains information about the game, including a list of socket connections to the clients in the game, the room code, the number of players playing the game, and their status (If they are active, if they disconnected, or if they finished the game)

Where does everything Run?

The Server is installed in the cloud using an Amazon Web Service. The server is always listening on a certain port, designated 10000 for this product. Any TCP connection requests on that port are accepted. The Client program is stored and run on the player's local device. The Server's domain address is hardcoded into the Client Program.

How does it work?

- Once the player runs the Client program, the socket attempts a TCP Connection to the Server. If the connection is successful, the client is presented with a TKinter GUI form, with options to create a game, or join a public/private game.
- JSON objects are used to transmit data between the Clients and the Server. Information such as picking a game to join, creating a new game, rolling a dice, moving a piece, the end of a turn, and chat messages are all sent to/from the server as JSON objects. As they have functionality to be converted to a string while retaining their data types, JSON allows the transmission of Integers, Booleans and Lists without worrying about conversion.
- The Server holds a list of current games and appends any new games a client creates. A new thread is created for each connection. If the client joins a game, the server checks the room code provided (if any) to see if they have access, or if the game is already started/full.
- Once there are sufficient clients in the game, they push 'Start Game'. Once the server verifies that all the players in this game have pushed the button, the game is launched, and the first player's turn begins. The timeout begins its countdown, and the first player has the opportunity to roll the dice.
- When the 'Roll' button is clicked, the client program verifies that it is actually their turn. If it passes, a JSON message is sent to the Server, who runs a random generator to produce a number between one and six. If the player's 4 pieces are in the home square, they require a six to be able to phase out. The client alerts the server if this is the case, and instead runs a "biased" dice, that instead gets a random number from one to eight. If the number is 6,7, or 8, it is considered a roll value of six (This gives players initially a 3/8 chance of rolling a six).
- The Server also uses a random generator to produce an integer between one and six to determine the Genie Status, which is changed depending on the integer returned. Once the Dice number and the Genie Status are calculated, they are inserted in a Dictionary that is JSON-standard, which is converted to a string and forwarded to all clients.
- Each JSON message has a (key, value) pair of the form ("colour": colour), which is used to identify the source (or if the Server created the message, the target) of the message. Once the client message receives the JSON message, it reconverts it from a string to a dictionary and uses a series of "if key in message" conditions to interpret the message. Depending on the message, the clients interface is updated to account for the new information (A piece is moved if "movement" is in the message, or a message is appended to the chatbox if "chat_msg" is in the message, for example.).
- Once a player rolled the dice and they got a result from the server, a function is called to see if any of the pieces are moveable, depending on the roll. If there are any, the piece(s) is highlighted on the board, and player can click on them to move the piece. If the Genie status in the player says you can take the genie, then the genie icon now appears in the player's home square.
- If a piece lands on the same position as another player's piece, a series of checks are done. If the opponent's piece just left their home space or is on one of the safe-spaces on the board, then nothing happens. If the opponent has the Genie, then the player's piece is killed, and sent back to the home space. Otherwise, the opponents piece is killed and returned to their home.
- Once a player has moved, and they have no additional turns, they send a message informing the server. The server calls a function to iterate through the list of connections, looping around if need be, finding the next client in the game that is still active.
- If a player disconnects or finishes the game, the Server alters the reference to their connection, so that it will no longer give them the Turn Token. The client displays a new

window, showing the scoreboard at the time you finished/disconnected. The result is different however, as those who finished the game still receive messages on the other players, so they can continue to observe the game. Those who disconnected are marked as such, and no longer are sent any messages by the server.