3 a. Provide a written response that does all three of the following:

Approx. 150 words (for all subparts of 3a combined)

i. Describes the overall purpose of the program.

This program is a basic choose your own adventure game, but it has multiple games, and is completely modular so it can be quickly modified to add or remove games. It allows for user inputs and user action to guide them through a set of predefined routes. The game output is completely dependent on the user inputs

ii. Describes what functionality of the program is demonstrated in the video.

In the video you can see the basics of the program running, you first see the program start and ask what game the user would like to play, then 0 is inputted, corresponding to the “Forest Adventure” game. Next, the 3 situations are run through and are given the required inputs to progress, all but the last being correct. This leads to a score of 2 and a game over message. The program ends with the user typing N when asked to play again

iii. Describes the input and output of the program demonstrated in the video.

The inputs are characterized by a question asking what the user would do in a situation, the user then inputs a letter corresponding to the option printed in the console. The program outputs whether the input is valid or not, and if it is correct or not. At the end of the game the program outputs the user’s score and either Game over or congratulations based on the score.

3 b. Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.

Approx. 200 words (for all subparts of 3b combined, exclusive of program code)

i. The first program code segment must show how data have been stored in the list.

```
// an array of games, each requiring the necessary gameObject interfaces
const gamesArray: gameObject[] = [forestAdventure, subwayFun];
// ____________________________________________________
```

ii. The second program code segment must show the data in the same list being used, such as creating new data from the existing data or accessing multiple

elements in the list, as part of fulfilling the program's purpose.

```
// runs for every item in the games array and prints out each game as an option
for (var x = 0; x < gamesArray.length; x++) {
    console.log(`${colorsArray[x]}${x}:`, gamesArray[x].gameName);
}
```

Then, provide a written response that does all three of the following:

iii. Identifies the name of the list being used in this response.

The name of the list in this response is “gamesArray” and it contains custom TypeScript interfaces, “gameObjects”

iv. Describes what the data contained in the list represent in your program.

Everything inside of the array contains gameObjects. These game objects represent the actual situations and messages that will run when a user plays a game. In essence, the array represents every game available to the user, and all playable games.

v. Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list.

The list manages a large amount of complexity, it is used to loop through and output choices for games to the user. Without the list, it wouldn’t be possible to loop through, and instead there would be a number of manually written console.log statements to print each choice to the user. Additionally, when the user actually selects a game, we are able to use the number to choose the game of the corresponding index in the list. This makes it even easier to modularize, instead of having to use a switch for every case of input.

3 c. Capture and paste two program code segments you developed during the administration of this task that contain a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

Approx. 200 words (for all subparts of 3c combined, exclusive of program code)

i. The first program code segment must be a student-developed procedure that:

- Defines the procedure's name and return type (if necessary)
- Contains and uses one or more parameters that have an effect on the functionality of the procedure
- Implements an algorithm that includes sequencing, selection and iteration

```
async function runGame(game: gameObject, lives?: number): Promise < number > {
    var gameScore = 0; // variable that stores the number of situations passed by the user
    var livesRemaining: number = 1; // default number of lives
    if(lives) livesRemaining = lives; // if lives are given give the user them
    console.log(game.welcomeMessage); // welcome the player to the game

    // run the start situation, to get the user started
    var startSituation = await runSituation(game.situationsObject.start);
    if (startSituation != true) livesRemaining -= 1;
    if(livesRemaining == 0) return gameScore;
    gameScore++; // increase the score by an increment of 1

    // this for loop will run and ITERATE for every situation within the situations object, and it will await user input
    for (var situation in game.situationsObject.situations) {
        if(lives) console.log(`You have ${livesRemaining} lives left`);
        let situationResult = await runSituation(game.situationsObject.situations[situation]); // wait for the runSituation program to complete
        if (situationResult != true) livesRemaining -= 1; // if the user gets an answer wrong, it will take a life away
        if(livesRemaining == 0) break; // if the user has no lives left, end the for loop, exiting the functions cycle
        gameScore++; // if they get the answer right, it will update the score and continue with the rest of the loop
        continue; // continues the loop
    };
    return gameScore; // return value of the runGame function, returns the number of correct answers from the user
};
```

ii. The second program code segment must show where your student-developed procedure is being called in your program.

```
var gameResults = await runGame(gamesArray[gameSelection]); // run the game and wait for it to stop
```

Then, provide a written response that does both of the following:

iii. Describes in general what the identified procedure does and how it contributes to the overall functionality of the program.

The identified procedure runs the game. It is called and awaited from the main procedure, this allows it to run all of the situation cases provided in the inputted game object. Every time the user gets an answer right, the score goes up, if the user is wrong, the loop is ended and the procedure returns the score.

iv. Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

The procedure is identified as “runGame” and it has a few key ideas. It has a return value of a Promise: number, it takes in as an input a gameObject and a number, lives.. The first in the sequence of the algorithm is creating a variable to store the user’s score, which starts off at 0. After this, we initialize the remainingLives variable, this is to make sure we can end the game when there’s no more lives left, if the lives input is provided, we let remainingLives equal it, otherwise we default it to 1 life.. Next, we show the user the gameObject’s welcome message, this is important to help the user know what the game is about. After this, we run the game’s starting situation using the runSituation procedure, which also takes in a situation interface from the gameObject; it is separate from the other situations for the sake of complexity. After awaiting the results of the runSituation for the start situation, we check if the user got it correct or not, as runSituation returns a boolean value (true = user got it correct, false = incorrect).. If they got it incorrect, we remove a life from the remainingLives, if there are no more lives remaining, we

return the gameScore and end the procedure, otherwise we continue with the program. If they got it correct, however, we increment the score by 1, and move onto the iteration. Now, we iterate through every situation in the situations list, awaiting for the runSituation procedure's results every time, every time we iterate, we use the same logic described earlier, with the returning the program if the user gets it incorrect, and continuing if they have it correct + adding the score. Once the user is done with all the situations, we return the score, the same as if they had lost, but with a higher score.

3 d. Provides a written response that does all three of the following:

Approx. 200 words (for all subparts of 3d combined)

i. Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First call:

```
var gameResults = await runGame(gamesArray[gameSelection], /* default 1 life */); // run the game and wait for it to stop
```

There can be many calls, but in this case, we'll assume the user inputs the number 0 as their input. This means runGame will be given the Forest Adventure gameObject.

Second call:

```
if ((await situationQuestion("Would you like to play again? Y/N")).toLowerCase() == "y") return runGame(gamesArray[gameSelection], 2); // await a user response to see if they would like to play again or not, if so, restart the main function to start the game again
```

ii. Describes what condition(s) is being tested by each call to the procedure.

Condition(s) tested by first call:

In the first call, we are not passing in any lives, so the default is 1 life in the procedure, this makes it harder for the user to pass the game, and also makes sure the lives aren't displayed to the user in the console, as there is a check in the algorithm that prints the remaining lives if lives are given to the program.

Condition(s) tested by second call:
In the second call, we are also testing the lives, but instead, we are given lives as an input for the procedure. This means the procedure is going to instead show the user the amount of lives remaining. This also means that the user will be able to survive longer than 1 incorrect answer.

iii. Identifies the result of each call.

Result of the first call:
The result of the first call is a terminal log that looks like:
You are in a Forest at night, you hear growling, its a bear! What do you do?
a: Run into a nearby den for the rest of the night
b: Run away from the noise until it is faint
Which option would you like to choose? a
You ran the wrong way! You got eaten by the bear :(
GAME OVER! You scored 0

Result of the second call:
The result of the second call is a terminal log that shows the user's lives if they get an answer wrong or right. Something that doesn't occur in the first terminal call:
You are in a Forest at night, you hear growling, its a bear! What do you do?
a: Run into a nearby den for the rest of the night
b: Run away from the noise until it is faint
Which option would you like to choose? b
You successfully avoided the bear! 🐻
You have 2 lives left