```
// index.ts
// Authors: [hidden as per CollegeBoard instructions]
// Date: 5/12/2021
// Sources: StackOverflow, TypeScript Documentation, Wikipedia ANSI Codes
// Known Bugs: No Currently known bugs
// Description: Choose your own game/adventure. You pick a game from a list, and are prompted
to make
// it through a list of "situations," if you get the answers right, you continue, otherwise the
game ends
// with the current score.

// Import the Readline module, built into Node.js
import * as readline from 'readline';


// Game essentials setup -- using this to setup the readline module and allow for user input in
the terminal

// TERMINAL SETUP ------------------------------
const rlInterface = readline.createInterface({
    input: process.stdin,
});

// functional component for the program, isn't meant to run games, but is meant to allow inputs
from users
var situationQuestion = function (question): Promise < string > {
    return new Promise < string > (function (res, rej) {
        process.stdout.write(`${white}${question} `);
        rlInterface.question("", function (answer) {
            res(answer);
        });
    });
};


// ATTRIBUTION: these ANSI color codes for terminal were found on
https://en.wikipedia.org/wiki/ANSI_escape_code#Colors
// and https://stackoverflow.com/questions/9781218/how-to-change-node-jss-console-font-color.
These color codes are
// not my own work
```

```
var magenta = "\x1b[35m";
var red = "\x1b[31m";
var green = "\x1b[32m";
var yellow = "\x1b[33m";
var blue = "\x1b[34m";
var cyan = "\x1b[36m";
var white = "\x1b[37m";

// array to make the colors easier to use later down the line
var colorsArray = [red, green, yellow, blue, magenta, cyan, white];
// ------------------------------------------

// used to calculate the amount of correct guesses the user has, conditionally.
const objectLength = function (object) {
    var size = 0;
    for (var key in object) {
        size++;
    };
    return size;
};

// INTERFACES ---------------------------------- (DEFINE ALL THE INTERFACES NEEDED TO HAVE A ROBUST STRONG-TYPED GAME SYSTEM)
// The Game Object stores the data for a certain game--the game settings and such--composed of multiple other types of TypeScript types
interface gameObject {
    gameName: string,
        welcomeMessage: string,
        situationsObject: situationsObject,
};

// Contains all the situations, also stores the starting situation to have a starting location for players
interface situationsObject {
    start: situation,
        situations: {
            [key: string]: situation
        },
};
```

```
// Interface for a specific situation, used to contain all the data needed that will be
processed later in the game
interface situation {
    conditional: string,
        deathMessage: string,
        survivalMessage: string,
        correctAnswer: string,
        options: object,
};

// GAME DEFINITIONS --------------------------
// Define the game "Forest Adventure" so it can be used in other places if ever needed, also
makes the array look cleaner
const forestAdventure: gameObject = {
    gameName: "Forest Adventure",
    welcomeMessage: "Welcome to the Forest Adventure, you have to make the right decisions to
survive being lost in the forest!",
    situationsObject: {
        start: {
            conditional: "You are in a Forest at night, you hear growling, its a bear! What do
you do?",
            deathMessage: "You ran the wrong way! You got eaten by the bear :(",
            survivalMessage: "You successfully avoided the bear! 🐻",
            correctAnswer: "b",
            options: {
                a: "Run into a nearby den for the rest of the night",
                b: "Run away from the noise until it is faint"
            }
        },
        situations: {
            1: {
                conditional: "Once you've made it far away from the bear, you see moss growing
on the trees! What's your next move?",
                deathMessage: "You ran the wrong way! The moss lead you deeper into the forest
🌿",
                survivalMessage: "You made it further outwards of the forest! 🌿🌿",
                correctAnswer: "a",
                options: {
                    a: "Run towards the direction the moss grows",
                    b: "Run in the opposite direction of the moss",
```

```
                },
            },
            2: {
                conditional: "You see a wrecked airplane on the right side of you, what do you
plan on doing?",
                deathMessage: "You lit the highly flammable jet fuel on fire and it exploded!",
                survivalMessage: "You successfully scavenged more supplies to survive!",
                correctAnswer: "c",
                options: {
                    a: "Try to start a fire with the fuel from the plane",
                    b: "Run towards the airplane to look for people",
                    c: "Go near the airplane carefully and look for supplies for survival"
                },
            },
        },

    },
};

const subwayFun: gameObject = {
    gameName: "Subway Fun",
    welcomeMessage: "It's time for some Subway Fun! Make the right decisions and make it to
your destination.",
    situationsObject: {
        start: {
            conditional: "You're making your way down to the New York Subway, it is extremely
crowded, and you see someone fall over in front of you, what do you do?",
            deathMessage: "You tripped and fell and also got hurt",
            survivalMessage: "You helped the person up and also successfully made it down the
stairs.. Nice Job!",
            correctAnswer: "b",
            options: {
                a: "Keep going forward",
                b: "Stop and help the person up, and then continue down the stairs"
            }
        },
        situations: {
            1: {
                conditional: "You're waiting for your train to approach the station, and
premilinarily step in front of the danger line, what do you do?",
```

```
                deathMessage: "You tripped and fell in front of the train",
                survivalMessage: "You safely waited for the train to arrive and go in safely!",
                correctAnswer: "b",
                options: {
                    a: "Keep going forward towards the train",
                    b: "Go backwards and wait for the train to safely approach the station,
then board it."
                }
            },
            2: {
                conditional: "The train starts moving and you're not holding onto anything",
                deathMessage: "You tripped in the middle of the train",
                survivalMessage: "You safely reached your destination!",
                correctAnswer: "a",
                options: {
                    a: "Find a seat or handle quickly",
                    b: "Hold on to your luggage"
                }
            }
        }
    }
}

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


async function main() {
    console.log(`${cyan}Welcome to Choose Your Own Game(Adventure)!`); // Welcome the player to
the game in the console/terminal
    console.log(`${white}It's time for you to pick a game!`); // Send a new message in terminal
for player to choose a game

    // 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);
    }
```

```
    const gameSelection = parseInt(await situationQuestion("Which game would you like to play? (Enter the number):")); // await user input on which game they want to play and turn it into an integer for later use
    if (!gamesArray[gameSelection]) { // Checks if the selected number by the user is valid
        console.log(`${red}Error loading, or the selection was invalid. Please try again.`); // if invalid, a new terminal message is printed and
        return main(); // the program restarts
    };
    console.log(`${green}Loading game "${gamesArray[gameSelection].gameName}"`); // show user that a game is being loaded
    var gameResults = await runGame(gamesArray[gameSelection]); // run the game and wait for it to stop
    if (gameResults == objectLength(gamesArray[gameSelection].situationsObject.situations) + 1 /* +1 is to account for the starting situation */ ) console.log(`Congratulations! You Passed all the Situations 🥳 (${gameResults})`) // if the user passes all of the situations congratulate them
    console.log(`GAME OVER! You scored ${gameResults}`); // show the user their results and score

    if ((await situationQuestion("Would you like to play again? Y/N")).toLowerCase() == "y") return main(); // 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
    rlInterface.close(); // close the opened terminal input session if the user doesn't want to play again
    process.exit(); // end the program if the user doesn't want to play again
};


async function runGame(game: gameObject): Promise < number > {

    var gameScore = 0; // variable that stores the number of situations passed by the user

    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) 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) {
        let situationResult = await runSituation(game.situationsObject.situations[situation]);
// wait for the runSituation program to complete, if user is correct, returns truthy
        if (situationResult != true) break; // if the user gets an answer wrong, it will end
the for loop and the game will end
        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
};

// this function is abstracted code, it runs the specified situation and returns a value of
true or false depending on if the user is correct or not
async function runSituation(situation: situation): Promise < boolean > { // returns as a
promise of boolean, takes in the situation parameter
    console.log(blue + situation.conditional); // print out the conditional statement that the
user has to answer
    for (var option in situation.options) { // run a loop to show all of the available options
to the user
        console.log(`${magenta}${option}: ${situation.options[option]}`) // print out all the
available options to the user
    }
    var userResponse = await situationQuestion("Which option would you like to choose?"); //
wait for a user response, run the situationQuestion function to do this
    if (userResponse.toLowerCase() == situation.correctAnswer) { // if the user's input is the
correct answer, run
        console.log(green + situation.survivalMessage); // print the message congratulating the
user for the right message
        return true; // if the user's response is equal to the correctAnswer, end the function
and return true
    }
    console.log(red + situation.deathMessage); // show the death message by default if the user
inputs the wrong answer
    return false; // return false by default if the true case is not met
};
```

```
// start the program
main();
```