Teaching fundamental computational thinking skills through a series of interactive computer games

Liam Aspell
17300046

Final Year Project – **2023**
B.Sc. Single Honours in
Computer Science/Computer Science and Software Engineering

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.

Single Honors in Computer Science and Software Engineering.

Supervisor: Thomas Naughton

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of Computer Science and Software Engineering qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Liam Aspell

Date: 27/3/2023

Liam Aspell

Acknowledgements

I would like to thank my supervisor Thomas Naughton for his contribution to this project, setting meetings consistently and offering advice and feedback where needed to allow me reach completion.

I would like to extend my thanks to my lecturers who played a crucial role in my education and gave me the help and guidance needed to get here.

I would also like to thank close family members and friends, along with my girlfriend Jennifer, who kept me motivated during my time completing this project, and throughout the academic year.

List of Figures

List of Tables

Abstract

Third Level Computer Science courses are amongst the courses with the highest dropout rates in the country, while the skills gained by students who complete these courses are highly in demand by employers in the Irish economy. It is essential to find a solution to this problem. An increase in exposure to computational thinking concepts from primary school onwards better equips our young people to develop their skills and possibly gain a passion for problem solving using computational thought. The Bebras challenge is a global initiative aimed at tackling this problem, by creating problems involving computational thinking to solve, and tailoring these challenges for a variety of different ages. CT Games is a framework for developing computational thinking games, playable in a web browser by children of a school going age. The games within this framework are designed to be interactive, graphically appealing, and to reward the knowledge of applying computational thinking skills to problem solve. The developed games are all based on Bebras tasks. The objective of this project is to develop 3 games on the framework, representing a selection of Bebras tasks. Each game must have a playable command line version, as well develop the game in the browser, aided using Scalable Vector Graphics. The games must scale in difficulty from easy early levels to more challenging later levels. The games must also be verified with the use of unit tests and user evaluation. It is hoped that as this reaches more students, it will enhance the computational thinking skills of future generations and solves many problems such as the ones stated.

Chapter One: Introduction

Summary

Chapter 1 describes a brief outline of the project, the motivation for this project and the solution which came about as a result.

1.1 Topic addressed in this project.

The topic at hand is to acknowledge the need for more resources in teaching fundamental computational thinking concepts to young children. This will be addressed in this project by the creation of computational thinking skills challenges, selected from the large collection of Bebras computational thinking tasks [1].

1.2 Motivation

We currently live in a world where technological advancements have become commonplace. The current state of technology is constantly evolving, and new areas, such as AI, Machine Learning and Neural Networks are consistently developing. It is predicted that these areas will have a significant impact on many different industries in the future, calling for the need for young people to have strong computational thinking skills to enhance this development. This is a catalyst to innovation and future advancement.

The CT Games project [2] aims to equip young students from a variety of age groups with a vital resource to develop their computational thinking skills, fundamental to their understanding of computer science and the topics directly related to it. Each of the games within the CT Games project tests the user's ability to put these skills into practice, by examining them in areas such as pattern recognition and decomposition, while also delivering this in an abstract, enjoyable manner. This allows users to have fun, but also to learn at the same time.

1.3 Problem statement

The goal of this project is to develop interactive games on the CT Games framework, which are designed to teach the users (Primary School Children) fundamental computational thinking skills to win the game. Several technical problems must be addressed to complete this. Firstly, a command line interface (CLI) version of the game must be completed, using Python [5], and incorporating the CT games framework boiler plate. Once this backend version has been completed, and rigorously tested by the developer using the command line, a front-end web application as an implementation of the command line game is then completed, which runs on the browser. This web application combines the underlying logic within the command line version, and pairs this with visual stimulants on the front-end to portray the relevant task and make it appealing and challenging to the end user.

1.4 Approach

The work undertaken to develop games on the CT games framework follows a logical process outlined in the CT games documentation. To build a game on the framework, the first requirement is to find a suitable Bebras task to replicate in the form of an interactive game. As part of this, thought must be given to the underlining logic within this task, and how this logic can be translated into Python code. Different variables which cover the question, the answer, and a suitable list of other (incorrect) answers must be constructed, and the explanation of how these different variables interact with each other make up the logic behind the games. From here, a basic cli running the backend Python code could be constructed. Throughout the development of the games, particularly my first game, I had access to a library of different games that were completed in recent years by students who have previously completed this project. I was able to play, test and examine games like the ones I intended to build, to give me ideas of how to implement areas, such as displaying SVG's on the frontend.

1.5 Metrics

Unit Testing

Unit Testing in Python is a suitable metric to verify the correctness of the code developed. My plan is to construct unit tests to verify each method making up the backend logic for each of the three games. Writing Unit Tests helps to ensure the quality of the code and will help to identify any issues with any of the methods developed, allowing me to rectify these problems before completion of the project.

User Evaluation

I plan to conduct a small phase of user evaluation at the completion of the code development, to gain an insight into the performance of the developed games, specifically to gather information about areas such as the easiness of understanding the problem posed in the game, how to reach the correct answer and the effectiveness of the front-end assets in displaying the problem instance. Overall, this feedback will be important in shaping the final games. I hope to also gain an insight into the age groups these games would be suitable for.

1.6 Project

This project helped me to develop skills including:

1. Sourcing and editing basic svg graphics for the frontend of games.
2. Pre-development planning of game logic.
3. Code development with Python programming language, particularly working with different data structures (tuples, lists etc.).
4. Refactoring and improving code for reusability and simplicity.
5. Source control commands for git and working with Gitlab.
6. Using built-in browser console and element inspection to debug code on the front-end.
7. Greatly improved my ability to interpret error messages and make changes based on this feedback, as well as using breakpoints, all of which speed up development.
8. Good practices relating to code testing and conducting user evaluation.

Chapter Two: Technical Background

Summary

The purpose of this chapter is to display an insight into the research behind the project and some of the technical material gathered.

2.1 Topic Material

Bebras is an international initiative aiming to promote initiative aiming to promote informatics and computational thinking among school students at all ages, The idea behind Bebras is to come up with solutions to problems provided using problem solving techniques that software engineers use to solve problems to tasks. Completion of Bebras tasks involves becoming familiar with a specific problem-solving methodology, and therefore attaining that skill to reference when tackling computational thinking problems in the future.

CT Games method to delivering a learning outcome differs slightly from the conventional Bebras tasks. The idea behind CT Games is to reinforce a computational thinking concept by *"presenting a student with completely different tasks in sequence allows one to develop new problem-solving skills. Our complementary view, effected with CT Games, is that repeating different instances of the same task can allow a student to practice and refine on a problem-solving strategy." [3]*

This emphasises the need for a game built on the framework to have an increase in difficulty as the game progresses. This can be achieved by incrementing the complexity of the games problem instance as the user reaches later levels, and by including randomness in the generation of these problem instances, to prohibit the possibility of achieving the correct answer using any method other than using the relevant computational thinking skill.

2.2 Technical Material

Brython

Brython [4] is an implementation of Python, used for client-side scripting on the browser. This project implements Brython to develop the methods and logic making up the backend of games on the CT Games framework. This Python code is interpreted by Brython to convert into scripts readable by a web browser.

Development of the games' code was done completely in Python. Existing knowledge of Python from previous projects undertaken as well as the official documentation, and resources such as stack overflow were indispensable throughout the development of the games. Being able to develop the games in Python will allow for the easy construction of unit tests, which will be explained in more detail later in this report.

CT Games Framework

CT Games Framework provided templates for representing Bebras tasks with the aid of Python. The framework provides game templates for the two different types of games, multiple choice answer games, and games that take a type as an answer, such as an integer or string. Using these templates and referencing the games built prior to my games, I was able to form ideas about how to write good code which represents the logic of the games I want to create. The framework provides methods for representation of the games on the frontend, and a web server which can be ran locally to view the games running on the browser. This is a valuable resource during the development process.

Inkscape

Inkscape [6] was used to create and modify SVG [7] images to go into the frontend web app, to create the visual representation of the game. Combining the set of utilities within Inkscape with the set of svg assets sourced from SVGRepo.com allowed me to refine graphical elements to fit into my games. This improves the front end of the game, making them more visually stimulating and appealing to end users.

Testing

Testing of the Python code is conducted in the form of unit tests which can be constructed using unittest [8], a unit testing framework which is part of the Python standard library. The general structure of created tests will include assertions to verify that methods, which take in a set of arguments and return a particular variable or an object, will consistently produce output which is correct to the specifications. Unittest cases will allow me to verify my games backend logic is free of bugs.

Chapter Three: The Problem

Summary

The purpose of this chapter is to clearly identify the user requirements, explain the technical problem and provide a deeper understanding of the two.

3.1.1 User Requirements

The User requirements for the construction of games on the CT Games framework are as follows:

1. Design computer games that each require a fundamental programming skill to win.
2. The games should be equipped with game instructions for both teachers and students, and these instructions should be clear and concise in explaining the functionality and object of each game.
3. The code developed during the construction of the games should be readable, understandable, easy to modify or add to, and covered with automated tests.
4. Games should be visually stimulating, appealing to end users by using Scalable Vector Graphics (SVG's) and can be enhanced using animations.
5. Games question instances must have a degree of randomness to minimize the possibility of two players in proximity being presented with the same question instance.
6. The games must contain multiple levels, each increasing the overall difficulty of the game instance compared to the previous one, to challenge the end user effectively.
7. For "Multiple Choice Question" games, the correct answer must not be obvious, or possible to be worked out using a method other than implementing the skill being taught by the game.

3.2 Problem analysis

The problem to be analysed in this section, is the selection of tasks which could be developed and represented as interactive games within the CT Games framework. These games must reflect a computational thinking concept and aid the users understanding of this concept through playing the game.

3.2.1 Christmas Trees

This objective of this game is to improve the computational thinking skill of pattern recognition. Pattern recognition refers to one's ability to identify patterns in a particular data set.

In this game, the player is presented with a sequence of Christmas trees with varying layers of decoration, namely the number of baubles on each tree. The objective of the game is the select the correct sequence of the trees from the least decorated to most decorated. The correct answer is selected from 3 possible multiple-choice answers. The game scales in difficulty by introducing a longer sequence of trees (4 trees in the early levels to 6 trees in the later levels). Increasing the difficulty reinforces the skill of pattern recognition examined by the game. The game teaches the user to take a random set of data and sort it in ascending order, in this case by using the example of baubles on a Christmas Tree.

3.2.2 Clearing Snow

Clearing Snow is a game that is designed to teach the computational thinking skills of decomposition and algorithmic problem solving. The user must first simplify the large problem into simpler ones and try to complete these smaller problems in sequence to beat the game.

The player is presented with a randomly generated, which is a 4x4 grid size in the first level of the game. Also observed is a snowplough, which is placed in a random position somewhere along the road within the grid. There are snowflakes scattered on all the roadways in the grid, informing the user that the objective is to clear this snow using the snowplough. From the ploughs starting position, the user must plot their way around the grid using the built in arrow system, to guide the snow plough into the desired direction, until they have navigated around all the roads in the grid, avoiding any squares in the grid without a road, including obstacles, while also not guiding the plough to move to a position outside out of bounds of the underlying array. To beat the game, the user must decompose the grid into smaller, manageable portions to construct the larger algorithm to guide the plough successfully, clearing all the snow in the process. This game demonstrates the nature of algorithms, as it takes a large problem and encourages the user to break this problem into smaller manageable chunks, to navigate successfully around the arena to clear all snow.

3.2.3 Glasses

Glasses is a game that reinforces the skills of pattern recognition and abstraction. The user is first presented with a set of rules (faces and glasses) and must infer from the rules to match a sequence of glasses (the answer) to a proposed sequence of faces without glasses, which make up the question.

The set of rules are displayed at the top level of the game question, and as previously mentioned contain a set of faces with different colour glasses attached to them. There is then another set of faces in the question area, not containing faces in the question area. This is the instance of faces to which the user must match faces to. The answer area contains 3 sets of glasses. Within the set, the correct answer contains the correct set of glasses to fit on all the correct faces, but with one pair of glasses not matching a face in the sequence. This pair is randomly allocated, and this is a manipulation of the correct answer that was generated by retrieving the values corresponding to the keys (faces). The other two incorrect answers are generated by taking the correct answer and replacing two pairs of glasses with incorrect pairs. This method of replacing values of the correct answer in this method allows for random generation of answers across all levels. Replacing only a small number of elements, 1 or 2 pairs of glasses each time, makes for a challenging experience, as the user is encouraged to investigate all answers as possible candidate solutions and contrast the quality of the solutions in comparison to the other answers that make the multiple choices.

Chapter Four: The Solution

Summary

4.1 Architectural Level

The architectural structure of the framework is consistent for all games, as they are all built through the CT Games framework. Most games that make up the framework follow the flow of first presenting the user with an image / sequence of images that make up the question, and this is usually complemented by a well-written explanation of the task. The user can then interact with the game by inputting their answer, such as multiple-choice answer, an input string along with others. This answer is then compared with the correct answer / set of correct answers for the game sequence, which is computed as part of the backend logic of the game. The outcome of this comparison will then determine if the user has correctly solved the problem instance. This applies for two of my games, 'Christmas Trees' and 'Glasses'. However, there is some disparity from the norm for the third game, 'Clearing Snow'. Clearing snow is a game that accepts multiple correct answers. In this game, there is an infinite number of correct answers for each game instance. This is to cater for the circular nature of the roads in the task, and the multiple different ways in which a user could solve the problem. It is also important to note that the user can move over a certain tile as many times as they want, subject to not hitting any tiles without roads or with obstacles and clear all the snow in on all the roads in the instance, they will beat the game, as their movements satisfy the criteria of the game. The nature of this game requires implementation of a method which calculates the correctness of a user's answer, rather than trying to match it to a predetermined answer. This required use of a unique function within the CT Games framework, known as process_input_and_solve_task(). This function, documented in the CT Games "Used when we need to process the players input to determine if it is correct or not" [9] reference for games where there are many possible correct solutions, e.g., programming style games) and/or where we need to compute the correct answer. Configuration of this function is explained further in 4.2.2.

4.2 Implementation

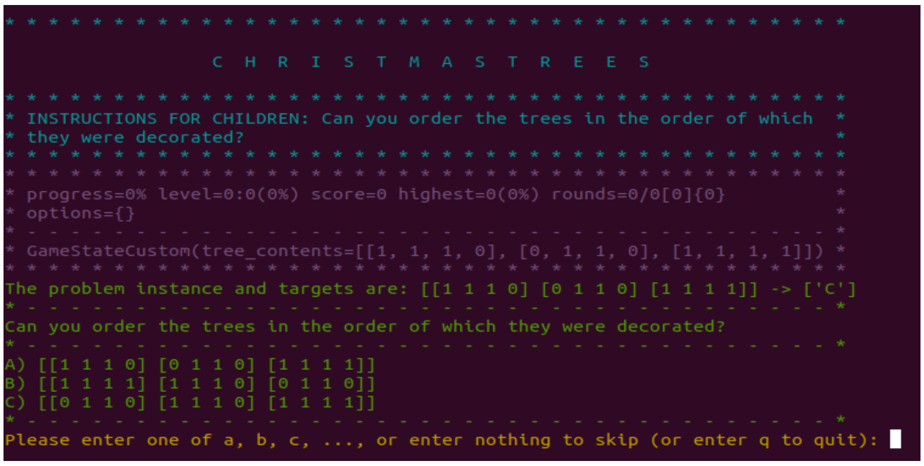
This section outlines the methodologies behind each of the games, and how these methodologies were implemented and developed to create each game.

4.2.1 Christmas Trees

Backend Logic

For my first game, Christmas Trees, construction of a command line interface (CLI) involving a list of lists, each list representing a tree, and each element in those lists containing a binary value (0 or 1), to represent the baubles. This list is constructed inside the build_tree_sequence () method, which takes in the number of trees as a variable and returns a list of lists named 'tree_sequence'. The 'tree sequence' variable represents the trees, and the length of each individual sub lists is equal to the number of maximum number of baubles on each tree. Each list contains a varying number of 1s (e.g, list 1 may contain four 1s, list 2 one 1, list 3 two 1s) and the correct answer is the same list but sorted into ascending order. To generate this correct answer, a get_correct_answer method implementing a bubble sorting algorithm is used to return the list in ascending form as specified. This algorithm sorted the lists based on their respective sums, as these figures would always be different because each tree was configured to have a different amount of decoration, ensured in the method 'build_tree_sequence'.

The variable which dictates the different levels is ‘number_of_trees’ in behaviour.py. As the user gets an increasing number of correct answers to questions and proceeds to later rounds, the number of trees will increase, as well as the number of baubles on these trees. This makes the game more challenging, as there are more factors to consider.



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                    C   H   R   I   S   T   M   A   S   T   R   E   E   S

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* INSTRUCTIONS FOR CHILDREN: Can you order the trees in the order of which  *
* they were decorated?                                                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* progress=0% level=0:0(0%) score=0 highest=0(0%) rounds=0/0[0]{0}          *
* options={}                                                                *
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
* GameStateCustom(tree_contents=[[1, 1, 1, 0], [0, 1, 1, 0], [1, 1, 1, 1]]) *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
The problem instance and targets are: [[1 1 1 0] [0 1 1 0] [1 1 1 1]] -> ['C']
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
Can you order the trees in the order of which they were decorated?
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
A) [[1 1 1 0] [0 1 1 0] [1 1 1 1]]
B) [[1 1 1 1] [1 1 1 0] [0 1 1 0]]
C) [[0 1 1 0] [1 1 1 0] [1 1 1 1]]
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
Please enter one of a, b, c, ..., or enter nothing to skip (or enter q to quit): █
```

Figure 1.1 CLI of Christmas Trees

Frontend Implementation

To create the front-end, a list of integer tuples represents the x and y positions of areas of the tree which a bauble would fit. This list of 12 values in length would be selected randomly when a particular game instance was presented, to ensure a random instance of the game was created each time. The positional values are complemented by a list of strings, each corresponding to the names of the 12 bauble svg files that are in the webapps image directory. This list is selected from at random, which allows each game instance to contain a unique sequence of random baubles, adding another factor of randomness. The randomness in the game ensures that there is a very low chance of two users sitting beside each other playing the same game would not receive an identical problem to solve.

Once a set of randomly selected positions and baubles are created, the front-end script displays the tree sequences that make up the question, the correct answer and the two incorrect answers. Firstly, question sequence is given a large size, the relative length of trees in the sequence is displayed. The baubles are then added on top of the tree SVGs, and list indexes make up multipliers to allow the script to place baubles on the correct position. *This can be seen in the*

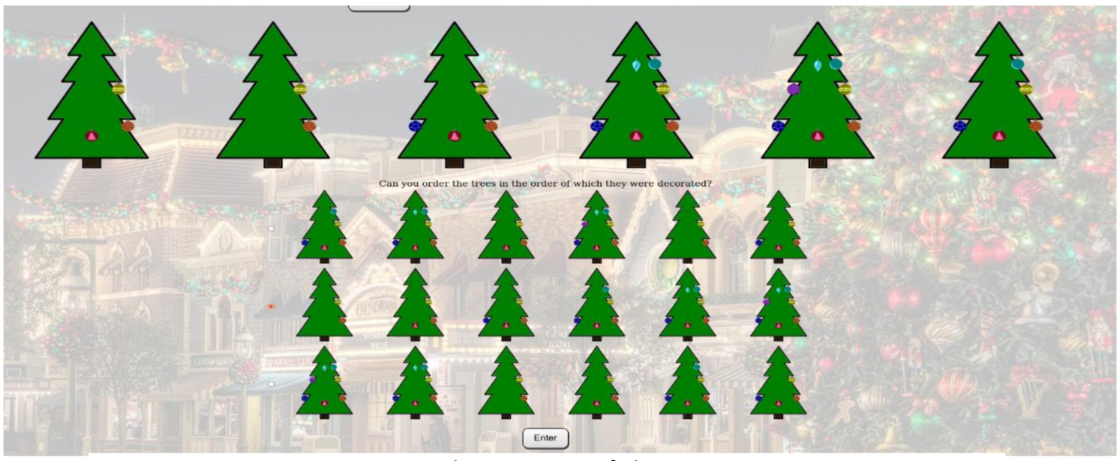

Can you order the trees in the order of which they were decorated?
Enter

Figure 1.2 Web App Version of Christmas Trees

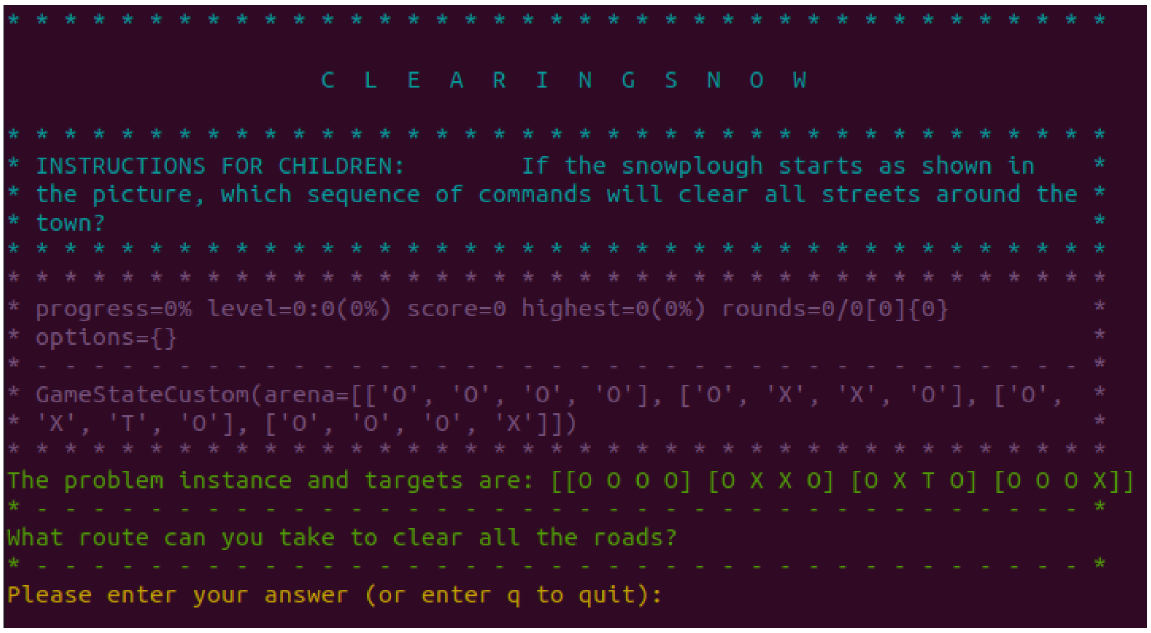
screenshot. A similar mechanism is used for displaying the baubles on the trees in the answer area, although all svg instances are scaled to be smaller to fit everything onto the screen.

4.2.2 Clearing Snow

Backend Logic

In Clearing Snow, a list of lists containing characters form a square array to determine the roadway for a particular question instance. A question instance is selected by randomly choosing a list from a set of preloaded lists for each level of the game. These are square lists and vary from a 4x4 array to a 6x6 array. An easy level of the game will utilise an arena the 4x4 array, whereas the larger arenas for level 2 and 3 would have 5x5 and 6x6 arrays returned respectively. The larger arrays require an increased number of operations to reach the correct answer, contributing to the increase hardness of the later rounds.

The games behaviour is controlled with variable known as ‘arena_size’. In the early rounds, this variable is assigned the value of 4. This is set to be incremented when the user completes 3 randomly generated arenas of each size, and on the successful completion of the 9th puzzle the game finishes.



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                    C  L  E  A  R  I  N  G  S  N  O  W

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* INSTRUCTIONS FOR CHILDREN:        If the snowplough starts as shown in     *
* the picture, which sequence of commands will clear all streets around the *
* town?                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* progress=0% level=0:0(0%) score=0 highest=0(0%) rounds=0/0[0]{0}          *
* options={}                                                                *
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
* GameStateCustom(arena=[['O', 'O', 'O', 'O'], ['O', 'X', 'X', 'O'], ['O',  *
* 'X', 'T', 'O'], ['O', 'O', 'O', 'X']])                                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
The problem instance and targets are: [[O O O O] [O X X O] [O X T O] [O O O X]]
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
What route can you take to clear all the roads?
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
Please enter your answer (or enter q to quit):
```

Figure 2.1 CLI of Clearing Snow

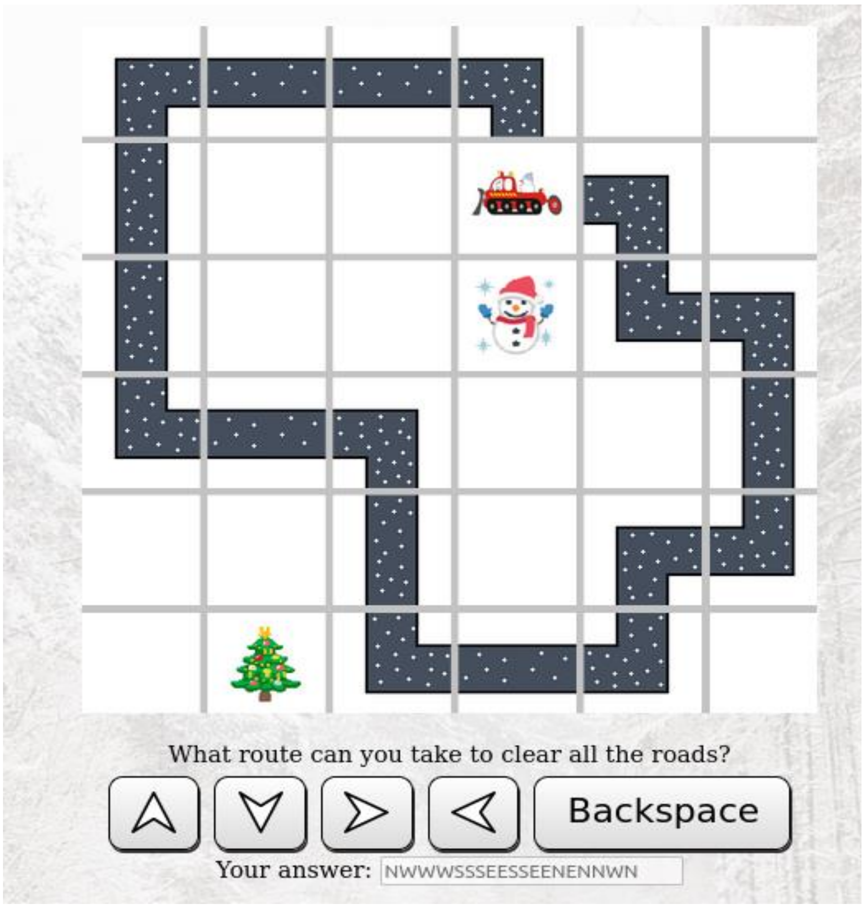
The correctness of an answer is determined by the outcome of the process_input_and_solve_task function, which takes in the game state (the randomly selected arena that for the user to solve) and the ‘player_input’ string, which is the set of values inputted by the user which correlate to movements of the plough around the grid. The function processes this input string, and stores updates to chars in the grid from ‘O’ (representing snow) to ‘_’ (representing cleared snow) to a variable ‘question_arena’ The function then calls upon ‘solve_arena’, again with the game_state variable passed, and returns the grid as it should look when all snow has been cleared (changes all ‘O’ to ‘_’). This solved arena is then compared against the resulting arena from inputting all operations, and if there is a match, the user has beaten this level and a notification stating this is returned. If the users answer fails to clear all snow, a subsequent message stating this is returned instead.

To ensure randomness of the tasks, a particular game level was selected from a collection 6 possible arenas for each level (4x4, 5x5, 6x6). Another layer of randomness is achieved through the placement of the snow plough. This is a randomised position for every instance of the game, and it is equally likely that it will be placed on any given square within the grid, granted that it is a square that previously contained a road. This allows for a different solution to be reached, if two users are playing the same arena at the same time. This is the desired level of randomness, as one user's solution will very rarely be a valid solution for another user's problem instance.

Frontend Implementation

The game is equipped with key value pairs to handle representation of the games problem instance on the frontend. Each of the keys in the set correlate to a possible backend arena that could be represented in the game_state variable. The frontend code firstly takes in this game_state arena and queries this against the data set to retrieve the corresponding front-end array which correlates to a sequence of chars which make up the arena by representing a square grid of SVG images. These SVG images grouped together make up a full roadway which is displayed as an instance of the problem.

The answer area is made up of 4 directional arrows as buttons, which input cardinal directions into a form. When the user is finished creating this answer, the sequence of directions correlates to the input string, representing the answer, which is passed to the back-end logic method


What route can you take to clear all the roads?

Your answer: NWWWSSSEESSEENENNWN

Figure 2.2 WebApp Version of Clearing Snow

‘process_input_and_solve_task’. The outcome of this method determines the correctness of the solution. If the solution is correct, a new task will be displayed, and these tasks will increase in complexity as the user continues to return correct answers.

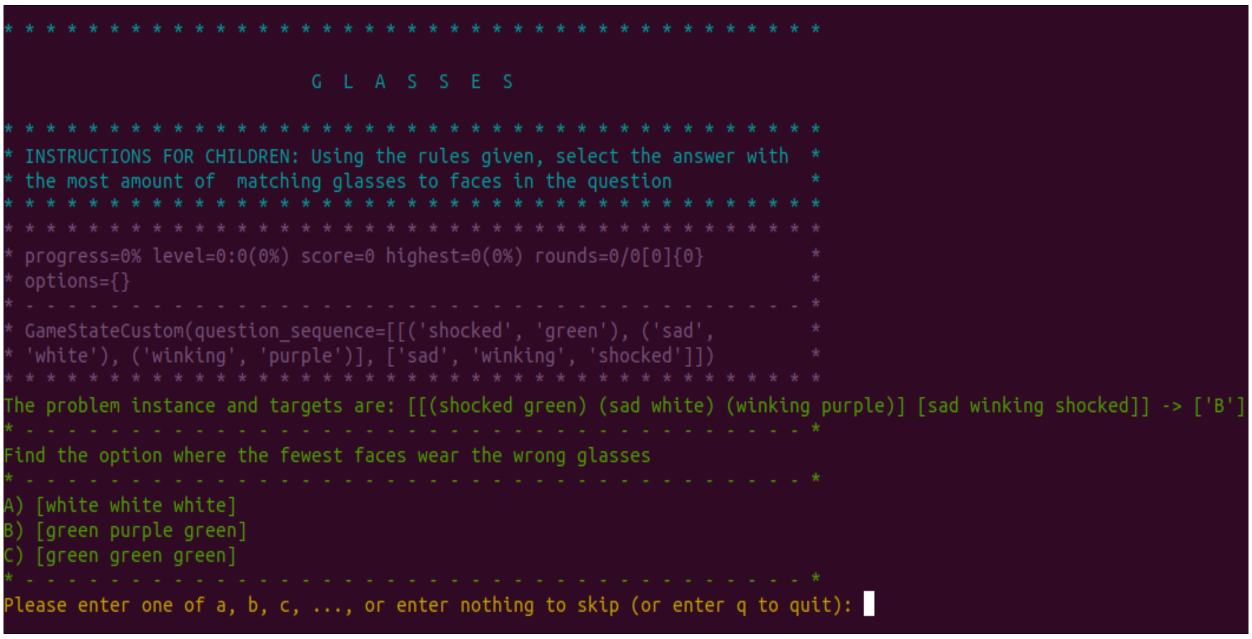
For additional clarity of the question instance, tiles containing roads, obstacles or blank space are separated out, and a large, blank SVG file coloured grey was added to allow the user to determine how far one input correlates to (e.g., inputting a W correlating to moving the plough one tile west). This was added as a response to user evaluation of the game and made the movements of the plough more obvious.

4.2.3 Glasses

Backend Logic

The game Glasses makes use of the data structure tuples, to represent the correspondences to faces and their respective glasses. This is the rule set that the user will be required to infer from to determine the correct answer. An example of a rule is (‘Happy’, ‘Blue’), which means that a happy face will wear blue glasses. Before the game instance is displayed, the key and value pairs a shuffled to create a randomized instance of the rules. This adds to the variety of the problem instances that could be displayed. Once this has completed, the rules are passed to the decide_on_problem_instance method which returns the rule set of the particular problem along with the question sequence combined in a list of lists structure. This list of lists along with the rule set is then used to generate the correct and incorrect answers respectively.

The game instances are determined by one rule variable, which determines the number of faces in the rules sequence. This is instantiated as ‘rules_amount’ and has a range of 3 to 5 depending on the difficulty of the problem, as it has a direct correlation to the number of unique faces displayed for a particular question. These faces all carry a unique set of glasses, making up a rule as mentioned earlier, to which the user is examined on. As the user completes instances of the game, this rule increments the number of faces to be displayed in the sequence. This makes the game scale upwards in difficulty as required.



```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                             G  L  A  S  S  E  S

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* INSTRUCTIONS FOR CHILDREN: Using the rules given, select the answer with  *
* the most amount of  matching glasses to faces in the question             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* progress=0% level=0:0(0%) score=0 highest=0(0%) rounds=0/0[0]{0}          *
* options={}                                                                *
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
* GameStateCustom(question_sequence=[[('shocked', 'green'), ('sad',         *
* 'white'), ('winking', 'purple')], ['sad', 'winking', 'shocked']])         *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
The problem instance and targets are: [[(shocked green) (sad white) (winking purple)] [sad winking shocked]] -> ['B']
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
Find the option where the fewest faces wear the wrong glasses
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
A) [white white white]
B) [green purple green]
C) [green green green]
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
Please enter one of a, b, c, ..., or enter nothing to skip (or enter q to quit):
```

Figure 3.1 CLI Version of Glasses

Frontend Implementation

The front end was built in a similar way to Christmas Trees. The set of rules, the question, and the three multiple choice questions were all requirements to display on the screen for the user to complete the problem instance. A large svg instance was created to store the rules combined with the question instance placed underneath, making up the question instance area. Like the method used to place baubles at a particular position, the code was configured to place the correct pair of glasses to the corresponding face, to match the underlying logic. This was achieved by using looping with indexes and a multiplier to calculate the correct co-ordinates of each pair, to allow it to be placed on the correct face.

The algorithm from above was replicated in the format_answer_area method, to space out the pairs of glasses correctly. The front-end script takes in the correct answer, and the two incorrect answers from the underlying logic of the game and displays these sequences of glasses in a uniform manner. I noted that I should add some text in later in development to emphasise the different aspects of the question to users who would not have been familiar with the game previously. This would also complement the game instructions well.

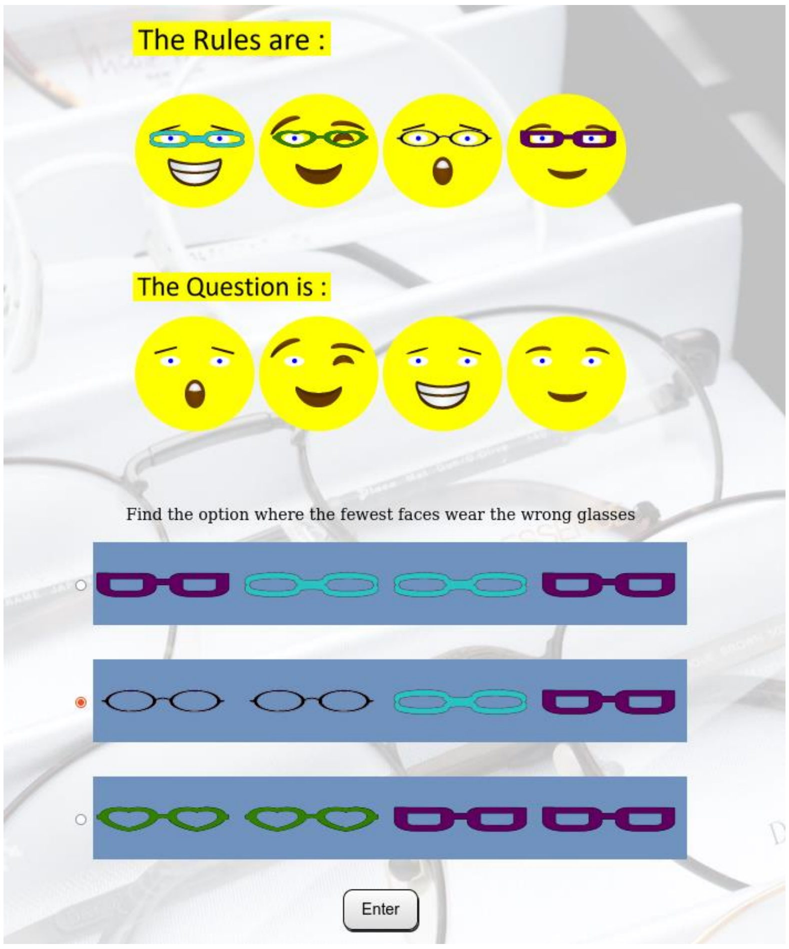

The Rules are :
The Question is :
Find the option where the fewest faces wear the wrong glasses
Enter

Figure 3.2 WebApp Version of Glasses

Chapter Five: Evaluation

Summary

This chapter describes in detail the different methods used to evaluate the games. The solution verification describes how I applied testing to the back end, in the form of unit tests implementing unittest. It also provides information on how web browser utilities and aided the verification of the front-end solution for each of the games. This chapter also describes the work around user evaluation, and how all these aspects helped to verify the games developed meet the desired specifications.

5.1 Unit Testing with Python

I built a suite of unit tests using unittest [8], for each of the games developed. This was essential to verify that the methods built to create the underlying logic for games worked as expected. As I have previously undertaken an internship in the area of quality assurance, unit testing with Python is an area that is particularly interesting to me. In the final weeks of this project, I aimed to build a unit testing suite for each game and populate it with various different tests that would verify a games correctness, and hopefully make it easier for me to spot some bugs. This was something I did not find replicated in other projects, but seemed straightforward once the units to be tested (i.e. the functions used in logic.py) for a particular game were imported.

Many of the unit tests follow a similar pattern. This is explained:

1. Import the necessary functionality to be tested at the beginning of the script (variables, functions)
2. Create some 'dummy' test data to aid the imported functionality.
3. Call a function and pass the relevant data, whether imported or created.
4. Evaluate the results of the function, in most cases run assert statements to ensure returned data is expected.
5. Verify the results with context to the overall game.

Tests on any function took place inside the function with the same name, with 'test_' attached to the beginning of it, inside the test class. This mapping kept everything in order, and keeping test scripts enclosed in functions ensured test results were brief.

Unit Testing ensured that all the small, simple functions that make up the games all work the way they are intended. Once I had verified each of these functions, I could determine that the underlying logic for each game was working successfully, without any unknown bugs. Usefulness of the scripts developed were evident in the later phases of code development. Testing played particularly significant role in Glasses. As it was the last of my games to be developed, I commenced developed of the unit tests while still developing the game. This meant that as I was creating the code that they acted as a baseline for the core functionality of the game.

If I was to do this project again from scratch, I would write test cases for a function upon completion of that function, so I could both verify that it works as expected, and to protect it from any future changes I make, which may contain errors, as unit tests would work to catch any deviation from the expected outcome, making it easier to detect any problems in develop, and thus making the development process more efficient. In the final few weeks, a shell script was constructed to kick off unit tests for the 3 games in a quicker, more efficient way, displaying the outcomes of all game's tests to one terminal. This mechanism contributed to the complete automation of the testing suite. I hope that building out unit tests for each function in a CT game will be replicated in future projects, to ensure the good standards of the games being developed, allowing the project to evolve over time.

5.2 User Evaluation

User Evaluation plays an important role in projects such as this one, where the goal is to create a useful, educational resource which is intended to be deployed in a school setting. User evaluation helps to prove the validity and viability of the games produced and assist me as a developer to highlight possible adjustments that could be made to further enhance the games. To do this, I requested a group of my peers to playtest each of the games, and then to answer a questionnaire for each of the games, analysing aspects from the clarity of the game's instructions, the initial difficulty of the games and the ability of the game to scale in difficulty in the later rounds as well as other areas such as the effect the images have, particularly to create an interesting game instance. The questionnaire provided for each of the three games are attached.

User Evaluation of the games allowed me to gather some useful information about the state of the games and use this feedback to carry out some changes to improve the games. One such example was found in the game Clearing Snow, where some users reported that the playing experience was slightly confusing. I understood that the ambiguous nature of being presented with a truck, a roadway and 4 navigation buttons, and the static nature of each of these elements, can make the game difficult to keep track of. Furthermore, The CT games framework does not really support updating of the front-end question arena in response to user input, before an answer is submitted. This proved problematic for me as the initial solution constructed was when the user chooses to navigate in a particular direction, they select their first direction button and the snow plough could instantly move to that position, and only submit the answer when they had navigated around the full roadway. As this would be quite an ambitious task that would have been quite time consuming, as well as making possibly the game too easy. I decided instead to revisit my game instructions to make them as clear as possible for future users. By using very clear, descriptive instructions which simulate a game instance, users are more likely to learn the task before playing and am hoping that the future users do not encounter any ambiguity with the task. If there is time at the end of the project, I want to investigate other ideas to further remove ambiguity in Clearing Snow.

From the user evaluation of the game Glasses, it was found to be most challenging game. Initially the game was to be simplified, where the correct answer would not contain any incorrect glasses in its sequence. This would mean that the player would have to find the answer where all glasses matched the correct sequence of glasses outlined in the rules. It was raised by my supervisor that this could make the game too easy, so I decided to carry out the necessary changes for it to conform exactly to the original Bebras task it was designed upon. Therefore, I would recognise this game to be suitable for an older age group, possibly for ages between 11-13. Although it was unexpected that it would be recognised as the most challenging, I feel it does a better job at demonstrating the computational thinking skill to the user.

Christmas Trees scored the best for aesthetics, it was pointed out that the combination of different colours leads to it being recognised as the most stimulating graphically, due to the different colours of the decorative baubles and their representations within the game. I feel this was achieved due to the use of bright and vibrant SVG files, along with a seasonal background. However, User

Evaluation also allowed me to uncover an issue relating to the game, where the user would get the correct answer by locating the tree with one bauble in the question sequence as the answer. This was a possible shortcut to getting the correct answer, which could be achieved without applying any computational thought to the task and was unseen in my own testing. I updated the game to ensure that each tree would have at least 2 baubles, as this would force the user to look more closely at the patterns in the multiple-choice answers and match them with the sequence to reach the correct answer.

5.3 Code Analysis with Pylint

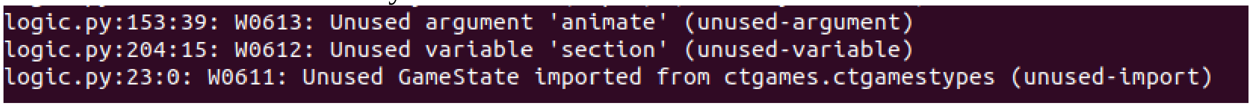
Pylint is a tool used to maintain good standards of Python code developed. An important requirement of games developed on the CT games framework is the necessity to build games that are readable and easy to modify or add to. Pylint is an important resource for Python projects such as this one to allow developed good achieve a good standard, by providing an automatic code analysis tool which gives the specified Python file a rating out of 10 based on its quality, as well as providing detailed feedback on how to improve this score. An example of feedback produced from Pylint analysis may be to remove a variable, that may have been passed to a particular method which was unused in the code block.

Pylint analysis was required for two files in each of my games: logic.py in the game directories top layer, and __init__.py, which is inside the webapp directory. Logic.py holds all the backend logic which forms the command line version of a game, while __init__.py holds front end functions for displaying the problem instances on the browser, formatting SVG elements into their correct positions and populating the answer areas. Running Pylint analysis on each of these files produced the following results.

Game: File	**Pylint Score**
Christmas Trees: logic.py	9.56
Christmas Trees: __init__.py	8.80
Clearing Snow: logic.py	9.38
Clearing Snow: __init__.py	9.32
Glasses: logic.py	9.82
Glasses: __init__.py	8.33
Average Score	9.2 / 10

Table 1 – Pylint Scores

As Pylint is a code analyser for native Python, it is not trained to understand Python code native to the CT Games framework, and therefore marks down any lines written containing properties related to the CT Games framework. This meant that it was impossible to achieve perfect (10/10) scores for the analysed files. Certain lines of code were flagged as possible defects, that were necessary to complete the games, as these lines contain code native to the framework. This is exemplified in screenshot, where the argument 'animate', which is native to the method 'Process_input_and_solve_task', and is required regardless of its use, is flagged as an unused argument. Such issues are unavoidable to keep the code functional for its desired purpose. . Due to the issues above with Pylint rejecting some code, I would assume this score to be higher also, if such a test provided more sophicasted analysis, that could recognise this code correctly. Pylint analysis was automated with a shell script to reach results quicker for all desired files as each file tested is in a different directory.



```
logic.py:153:39: W0613: Unused argument 'animate' (unused-argument)
logic.py:204:15: W0612: Unused variable 'section' (unused-variable)
logic.py:23:0: W0611: Unused GameState imported from ctgames.ctgamestypes (unused-import)
```

Figure 4 Some feedback from Pylint analysis that could not be corrected

To conclude, each of the files tested achieved high scores, scoring a 9.2/10 on average. This means that the code has good readability, hopefully making it easier for anyone who wishes to enhance or modify any of the games in the future, while also making the code less likely to contain any bugs.

Chapter Six: Conclusion

Summary

This chapter reflects on the project, the contribution to the state-of-the-art and future work that would enhance the games completed in the project.

6.1 Contribution to the state-of-the-art

By completing the project, I have added 3 more playable games into the CT Games library, which will go on to be distributed to primary schools across the country, with the intention of providing an important resource for school children looking to gain an insight into computational thinking, putting their skills to the test. The games aim to teach a variety of skills and require problem solving using many different methodologies. I hope that the variety in the playable games provides a comprehensive overview of some of the key concepts behind computational thinking, and that upon playing and winning the games that users become comfortable with these diverse and challenging concepts.

6.2 Project Approach

This project relied heavily on learning the theory behind concepts needed to build the games, followed by implementing these concepts to create the games. In a way, this mimics the style of the CT games produced, in which a user familiarizes themselves with the concept and is then examined on it. The CT games framework ties different aspects of development together in a unique way, by providing a backend to design and develop the underlying game, and then allowing the developer to represent this logic in an appealing and interactive manner on the front-end. To build a game on this framework required theoretical knowledge in areas such as developing and debugging console applications with Python, how to use data structures to represent game instances and how to manipulate these data structures to generate correct / incorrect answers. Overall, I found the development phase of the project to be quite slow initially, as learning the framework and technologies involved more time consuming to learn and implement on a practical level. However, upon completion of the first game, it became much easier to develop the second and third games, as the framework became less ambiguous with some experience. A comprehensive plan of the entire project can be seen below.

The plan of the project was as follows:

1. Research the project, play some games on the framework, and set up development utilities (Virtual Machine, Git, SSH, PyCharm Ide) (18/10/2022)
2. Commence development of the 1st game.
    - Design the games logic (Question, target answer, instance construction methods etc.) (31/10/2022)
    - Construct an appropriate working CLI version of the game. (6/11/2022)
    - Design / Source SVG assets to make up the front end of the game. (12/11/2022)
    - Pair the assets from above to the underlying logic to complete the front-end. (23/11/2022)
    - Develop the instructions, facts, and other text to improve the game experience. (9/12/2022)
3. Develop the 2nd game.

    - Design the games logic (Question, target answer, instance construction methods etc.) (16/12/2022)
    - Construct an appropriate working CLI version of the game. (23/12/2022)
    - Design / Source SVG assets to make up the front end of the game. (14/1/2023)
    - Pair the assets from above to the underlying logic to complete the front-end. (22/1/2023)
    - Develop the instructions, facts, and other text to improve the game experience. (29/1/2023)
4. Develop the 3rd game.
    - Develop the games logic (Question, target answer, instance construction methods etc.) (7/2/2023)
    - Construct an appropriate working CLI version of the game. (14/2/2023)
    - Design / Source SVG assets to make up the front end of the game. (21/2/2023)
    - Pair the assets from above to the underlying logic to complete the front-end. (28/2/2023)
    - Develop the instructions, facts, and other text to improve the game experience. (1/3/2023)
5. Conduct testing of the developed games to verify the code developed. (15/2/2023)
6. Carry out a phase of user evaluation and document the results. (22/3/2023)

6.3 User Requirements Reflection

Upon completion of the development and testing of the games, it is important to reflect on the user requirements laid out in the initially for the project. The initial set of user requirements outlined in chapter 2 were revisited at the completion of each game, to ensure each game was up to the standard of an appropriate game in this project. Below is the original list of requirements, and a checklist for each game developed.

User Requirement	Christmas Trees	Clearing Snow	Glasses
Design computer games that each require a fundamental programming skill to win.	✓	✓	✓
The games should be equipped with game instructions for both teachers and students, and these instructions should be clear and concise in explaining the functionality and object of each game.	✓	✓	✓
The code developed during the construction of the games should be readable, understandable, easy to modify or add to and covered with automated tests.	✓	✓	✓
Games should be visually stimulating, appealing to end users by using Scalable Vector Graphics (SVG's) and can be enhanced using animations.	✓	✓	✓
Games question instances must have a degree of randomness, to minimize the possibility of two players in proximity being presented with the same question instance.	✓	✓	✓
The games must contain multiple levels, each increasing the overall difficulty of the game instance compared to the previous one, to challenge the end user effectively.	✓	✓	✓
For MCQ games, the correct answer must not be obvious, or possible to be worked out using a method other than implementing the skill being taught by the game.	✓	Not relevant as Clearing Snow isn't an MCQ game	✓

Table 2 – User Requirements Criteria

Each of the games is configured in a manner that meets the minimum user requirements to be a valid addition to the CT games framework. Although there is always scope for improvements upon each of the games, I feel the games were each developed to an acceptable standard relative to the amount of time given to each aspect their development. Possible improvements to the games are described further in the next chapter.

6.4 What I would do with more time.

On the project level, the future work is to design and develop more games to run on the CT Games framework, each unique in their approach to delivering a suitable learning outcome relating in some way to the concepts underlying computational thought, using important references such as examples of Bebras tasks.

For the games I have developed, I would like to enhance the games further by adding animations to improve the overall aesthetic to the games, making them more interactive and appealing to students. One animation that I wanted to complete was the movement of the snow plough around the arena for clearing snow, after the user has inputted their answer, that would clear up the ambiguity seen in the user evaluation earlier, for new players who are unsure of how the game works. To do this, I would first allow the user to input their answer, and then simulate their answer on the roadway given in the question, to allow them to see if they have made a mistake somewhere in their solution. This would work in tandem with the logic method which takes in the users input and returns a correct message if the user clears all the snow.

If the opportunity ever arose, I would enjoy the chance to develop more games on the framework. The concept of computational thought is something that needs to be integrated in students' educations from the outset. To do this successfully, many resources are needed to reinforce the skills involved. It is a project that I hope will grow with more variety of games in the future and resources such as this one become integral to education in the future.

References

[1] https://www.bebras.org/goodtask.html. Bebras Tasks. Accessed October 31st, 2022.

[2] https://ctgames.readthedocs.io/en/latest/. CT Games. Accessed October 20th, 2022.

[3] https://ctgames.readthedocs.io/en/latest/introduction/about.html. CT Games ‘About’ page. Accessed November 12th.

[4] https://brython.info/static_doc/en/intro.html. Brython Project Documentation. Accessed October 20th, 2022.

[5] https://www.python.org/about/. Python Documentation Site. Accessed October 15th, 2022.

[6] https://inkscape.org/. Inkscape Site. Accessed November 14th, 2022.

[7] https://www.w3schools.com/graphics/svg_intro.asp. SVG information page. Accessed November 14th, 2022.

[8] https://docs.python.org/3/library/unittest.html. Python Unittest Documentation. Accessed February 15th, 2023.

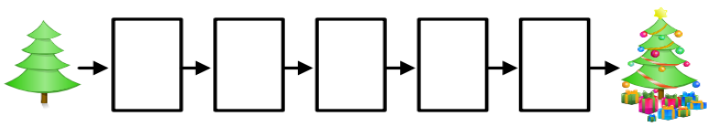
[9]https://ctgames.readthedocs.io/en/latest/command_line_games/function_process_input_and_solve_task.html?highlight=process_input_and_solve_task%20. CT Games documentation for method. Accessed December 1st, 2022.

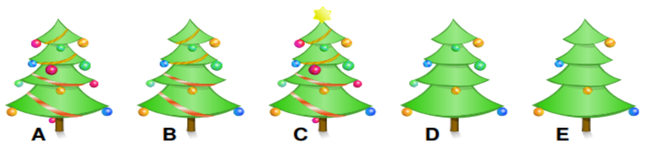
Appendices

Original Bebras Tasks (Inspiration for Games)

Christmas tree

Taro the Elf is planning an animation made from a sequence of pictures of a Christmas tree. The animation starts with no decorations, and in each picture more decorations are added. Unfortunately, the pictures got mixed up! Now Taro must find the correct order again. Luckily, he knows which pictures are first and last.





QUESTION

What is correct order of the pictures?

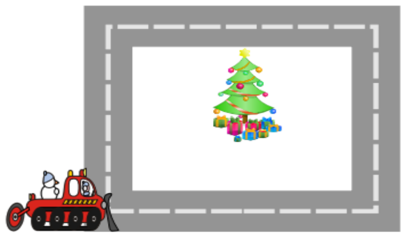
A. D E B A C
B. E D A B C
C. D E A B C
D. E D B A C

Clearing snow

The streets around the town square have to be cleared of snow before the Christmas market opens. Each side of the square is 100 metres long.

A remote-controlled snowplough can be controlled by the following commands:

- F: The snowplough moves forward 100 meters.
- R: The snowplough turns right.
- L: The snowplough turns left.



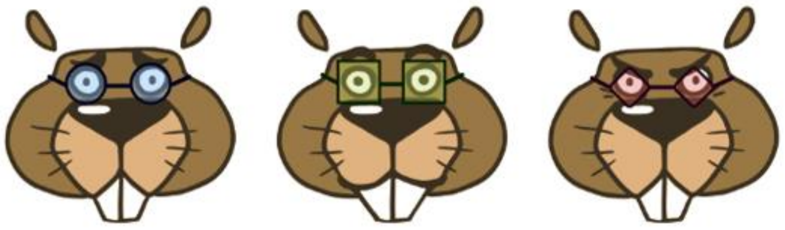
QUESTION

If the snowplough starts as shown in the picture, which sequence of commands will clear all four streets around the town square?

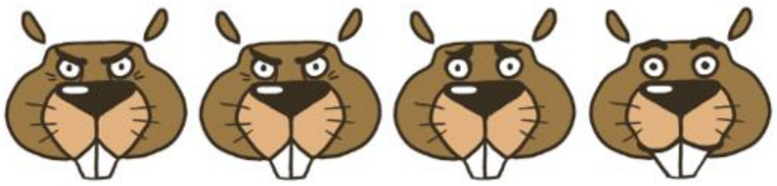
A.	B.	C.	D.
F	F	F	F
R	L	L	R
F	F	F	F
R	R	L	L
F	F	F	F
R	L	L	R
F	F	F	F

Faces and Glasses

Different facial expressions match different styles of glasses as shown here.

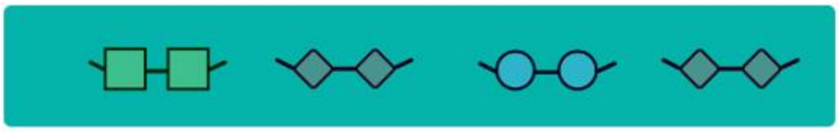


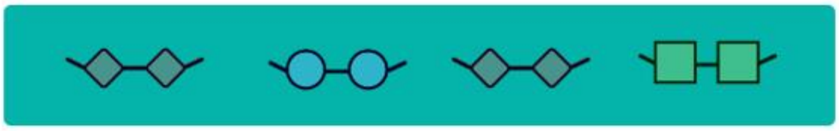
Here is a new sequence of faces:



Four options of glasses for this sequence of faces are given below.
Each option lists one style of glasses for each facial expression in order from left to right.

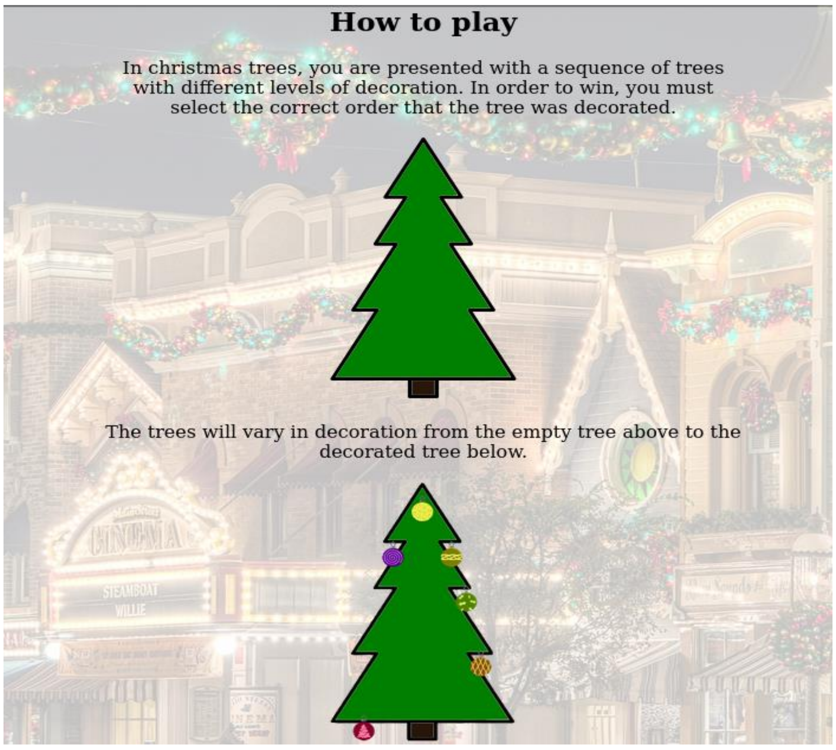
Question:
Find the option where the fewest faces wear the wrong glasses.





Game Instructions

Christmas Trees

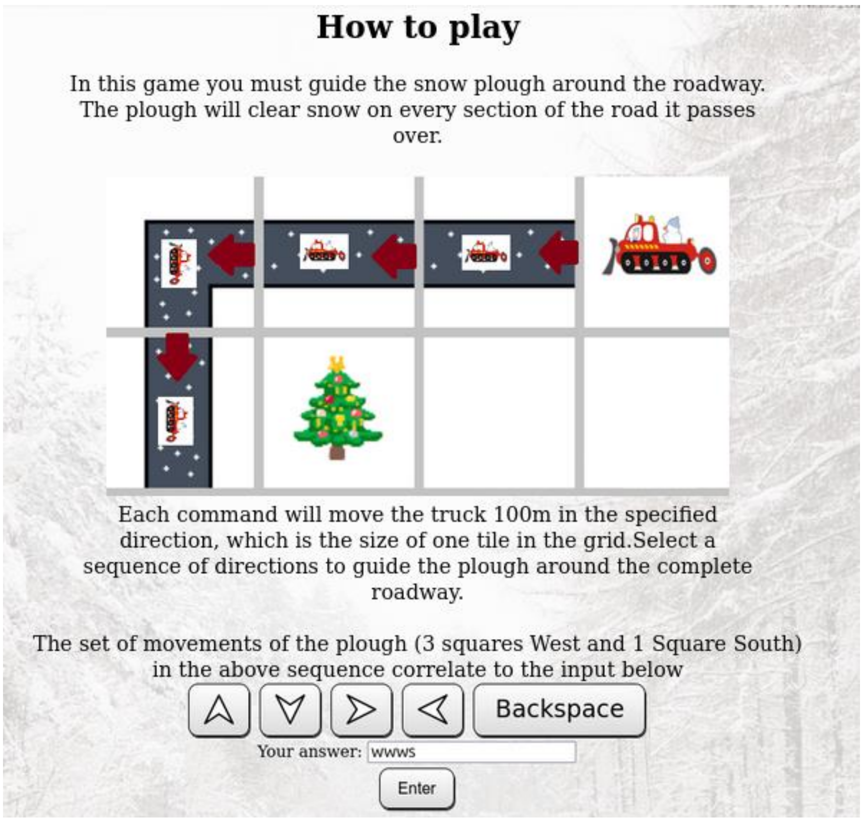


How to play

In christmas trees, you are presented with a sequence of trees with different levels of decoration. In order to win, you must select the correct order that the tree was decorated.

The trees will vary in decoration from the empty tree above to the decorated tree below.

Clearing Snow



How to play

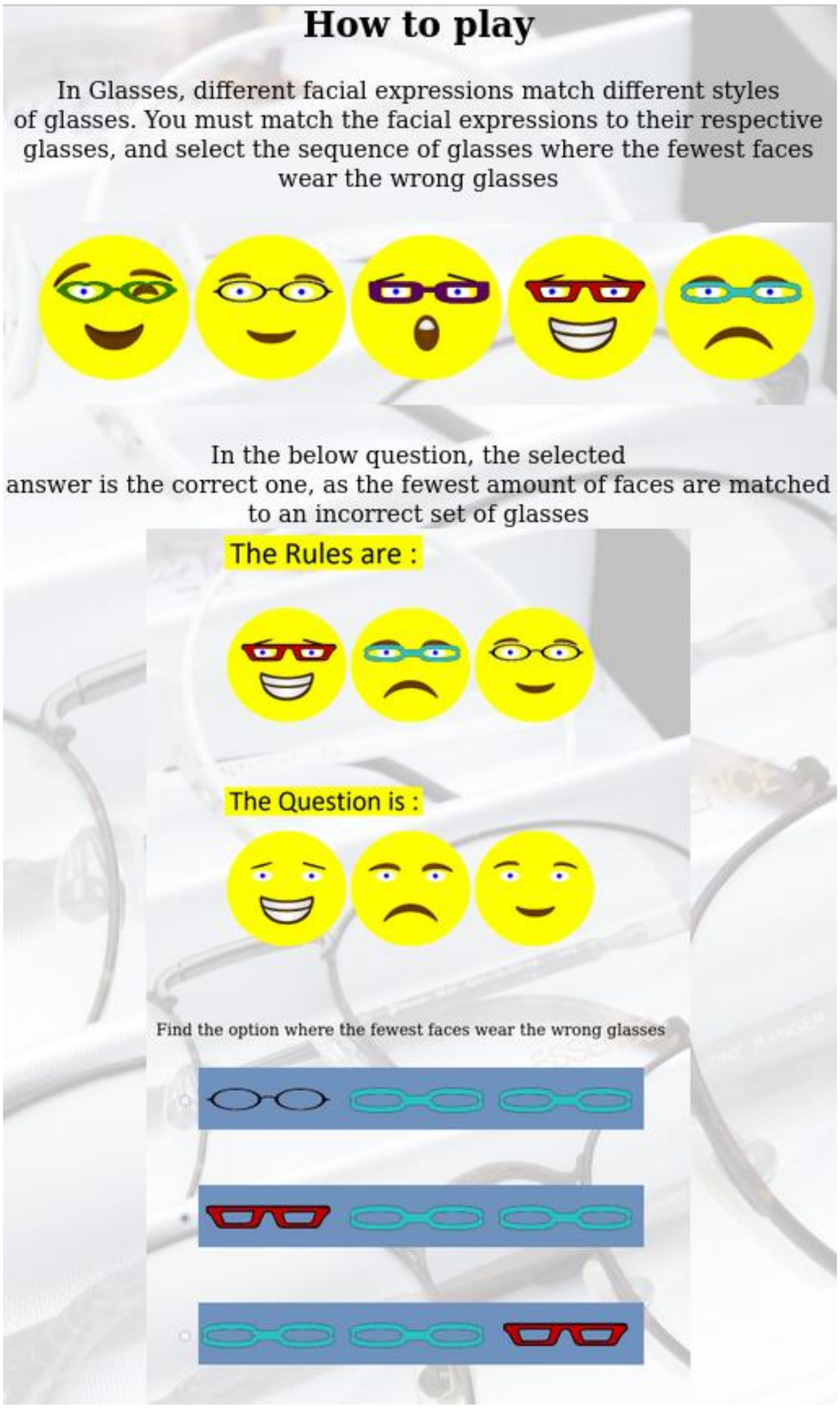
In this game you must guide the snow plough around the roadway. The plough will clear snow on every section of the road it passes over.

Each command will move the truck 100m in the specified direction, which is the size of one tile in the grid.Select a sequence of directions to guide the plough around the complete roadway.

The set of movements of the plough (3 squares West and 1 Square South) in the above sequence correlate to the input below

Your answer: wwws

Glasses


How to play

In Glasses, different facial expressions match different styles of glasses. You must match the facial expressions to their respective glasses, and select the sequence of glasses where the fewest faces wear the wrong glasses

In the below question, the selected answer is the correct one, as the fewest amount of faces are matched to an incorrect set of glasses

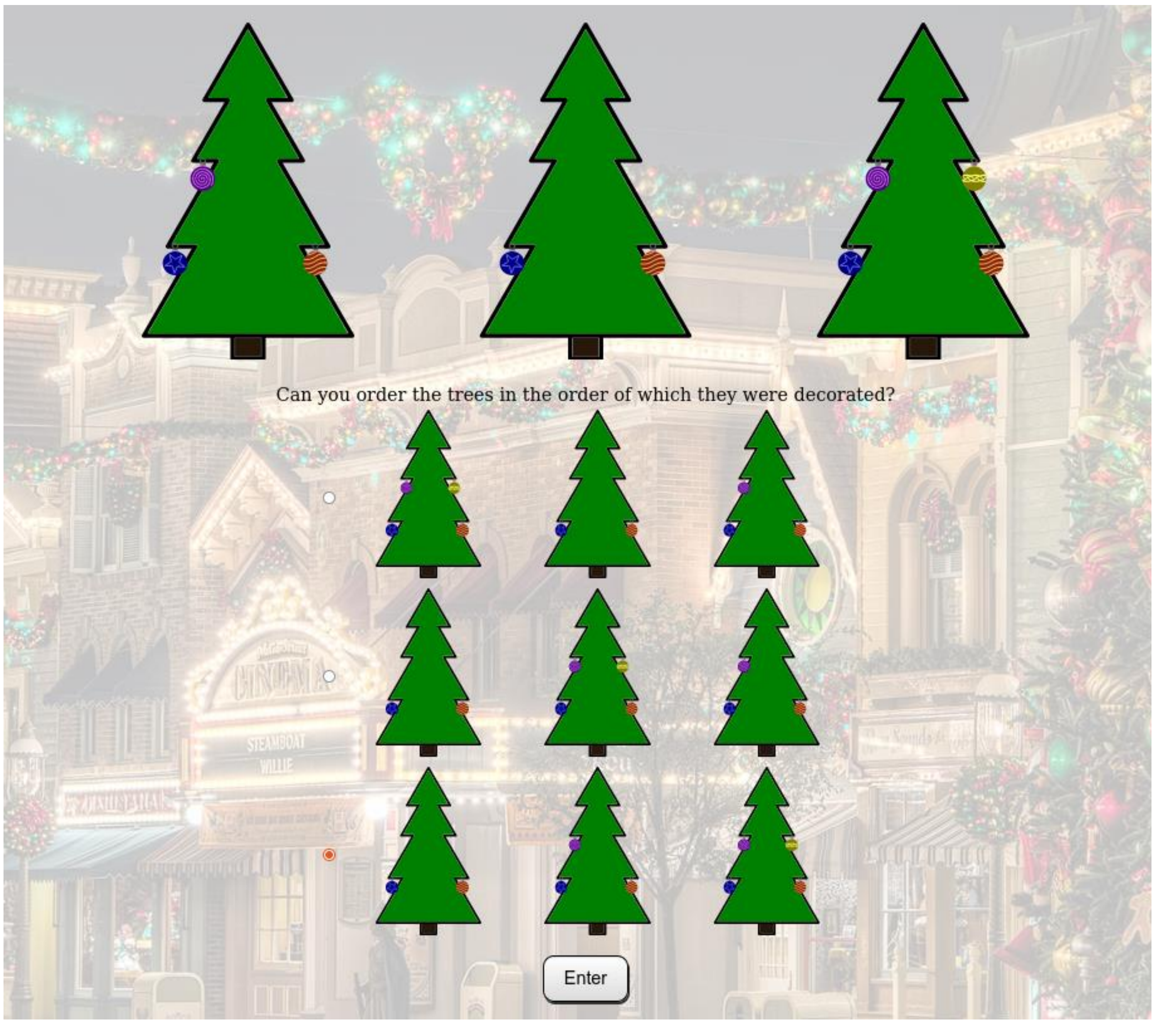
The Rules are :

The Question is :

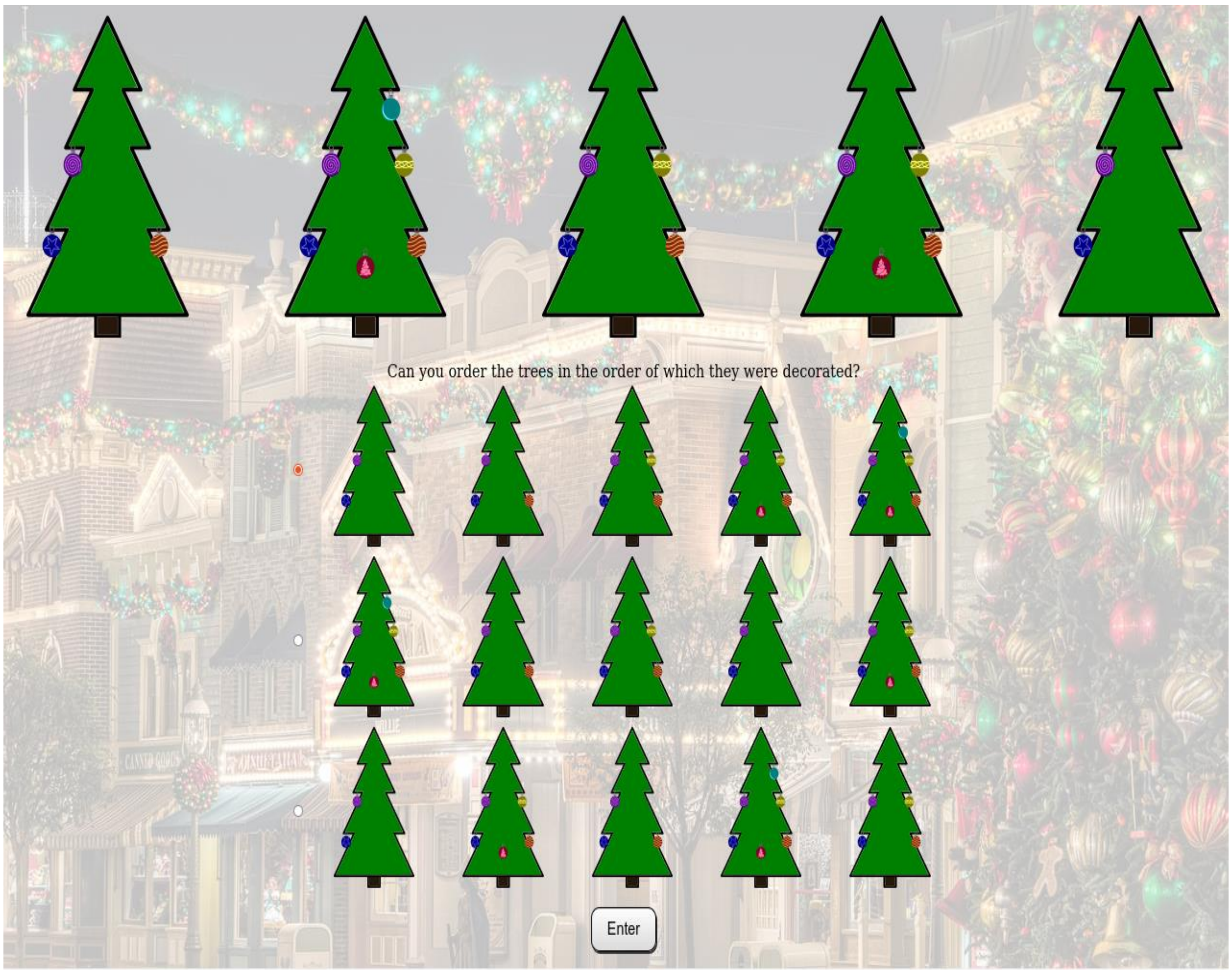
Find the option where the fewest faces wear the wrong glasses

Easy, Medium, Hard Levels

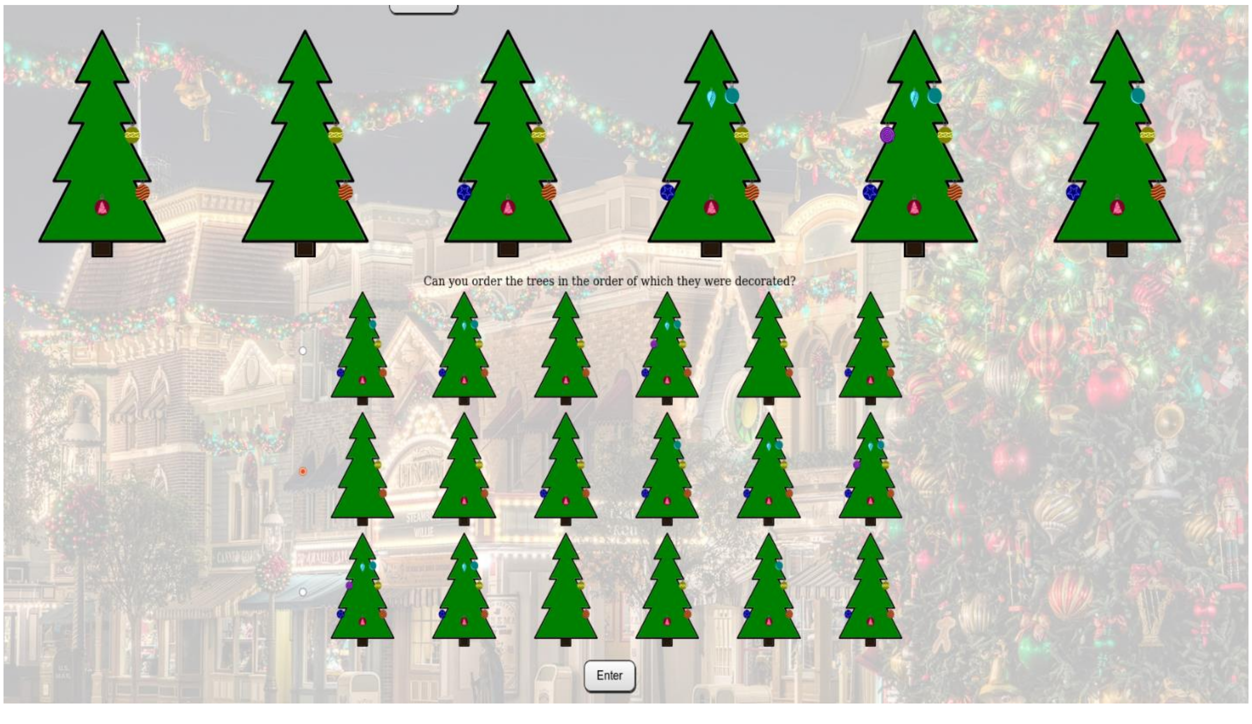
Christmas Trees – An Easy level


Can you order the trees in the order of which they were decorated?

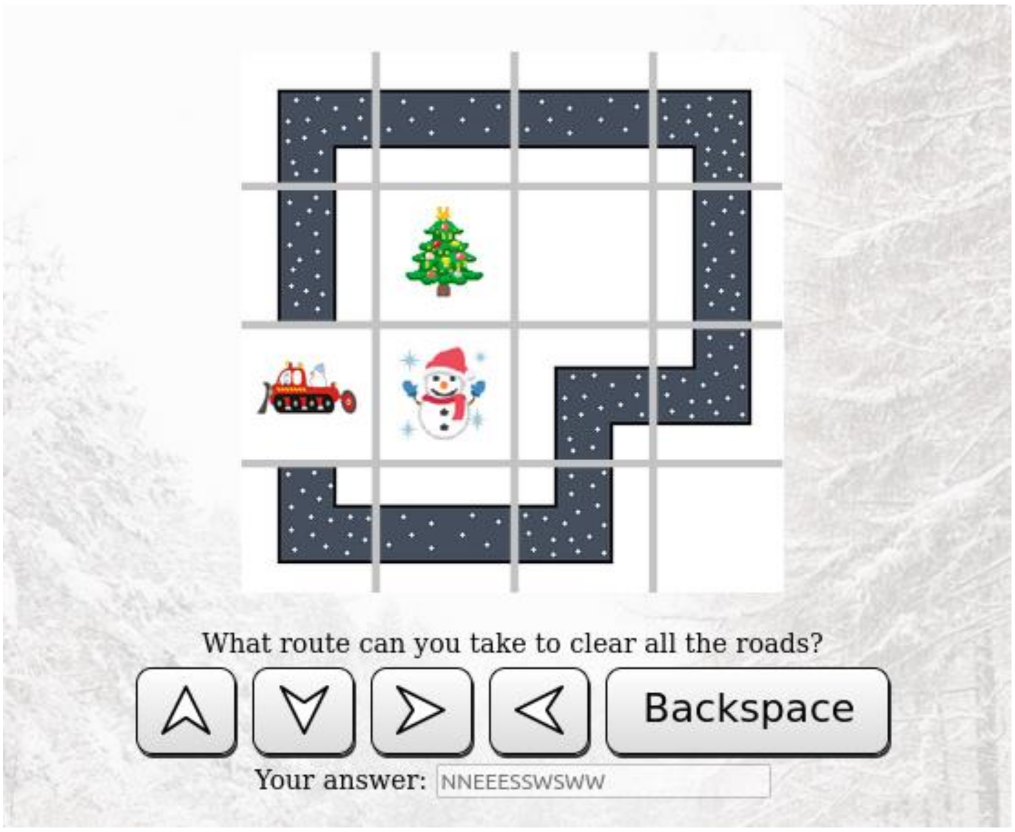
Christmas Trees – A Medium level


Can you order the trees in the order of which they were decorated?

Christmas Trees – A Hard level

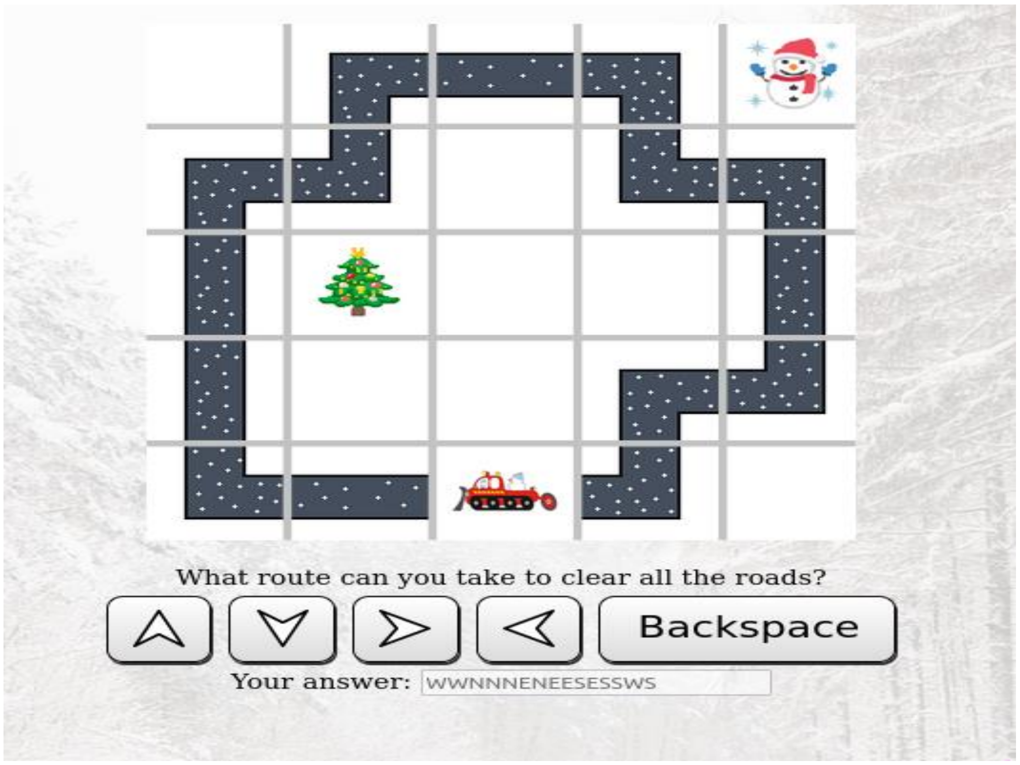

Can you order the trees in the order of which they were decorated?

Clearing Snow – An Easy level


What route can you take to clear all the roads?

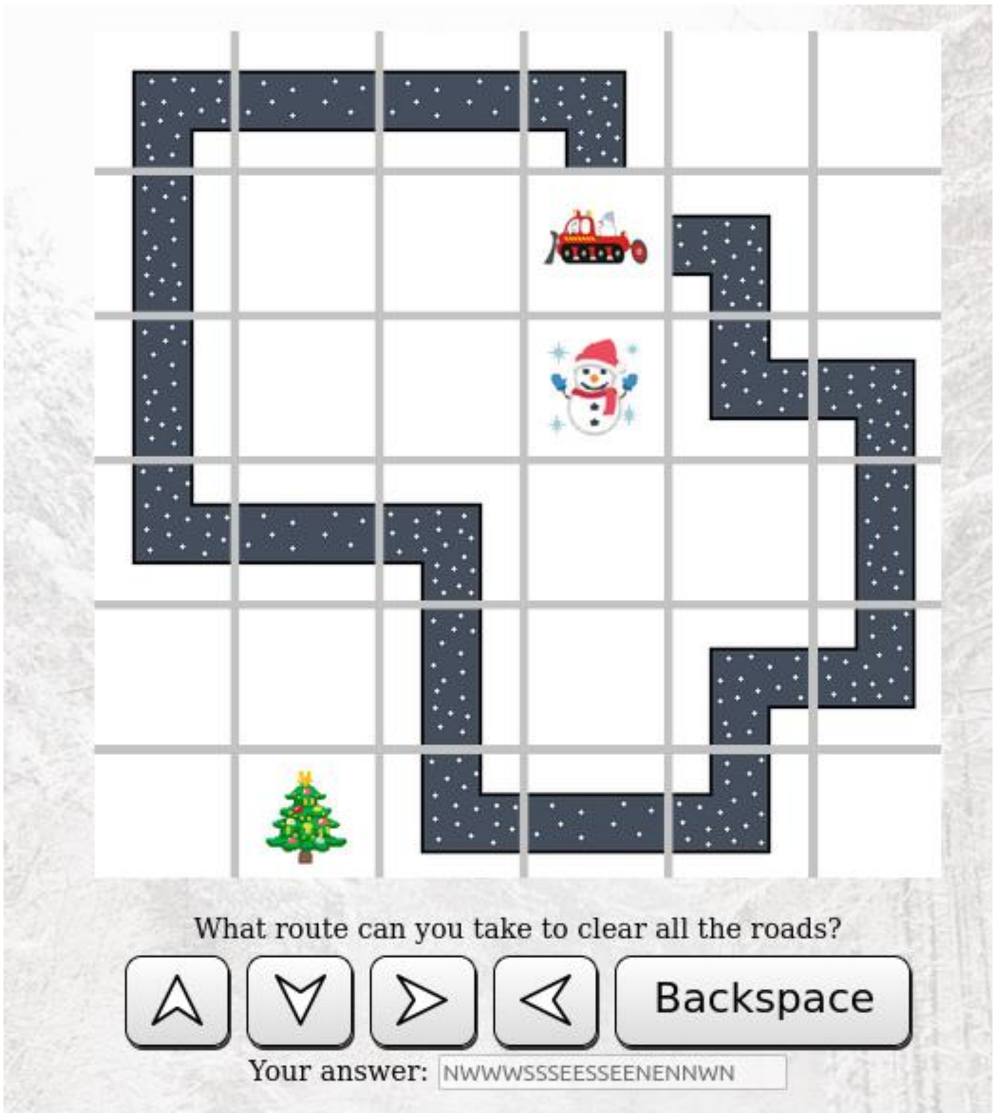
Your answer: NNEEESSWSWW

Clearing Snow – A Medium level


What route can you take to clear all the roads?

Your answer: WWNNNENEESESSWS

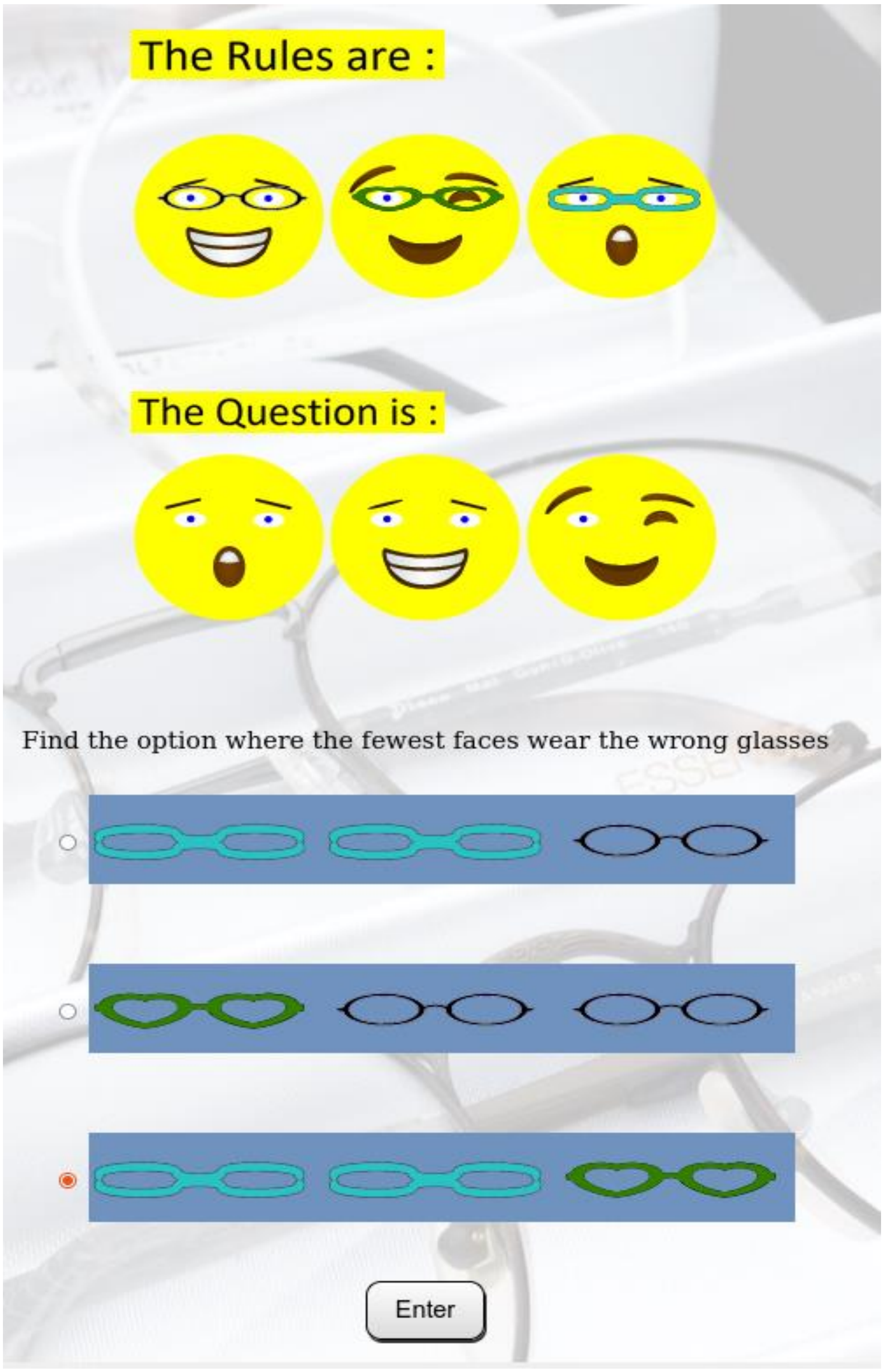
Clearing Snow – A Hard level



What route can you take to clear all the roads?

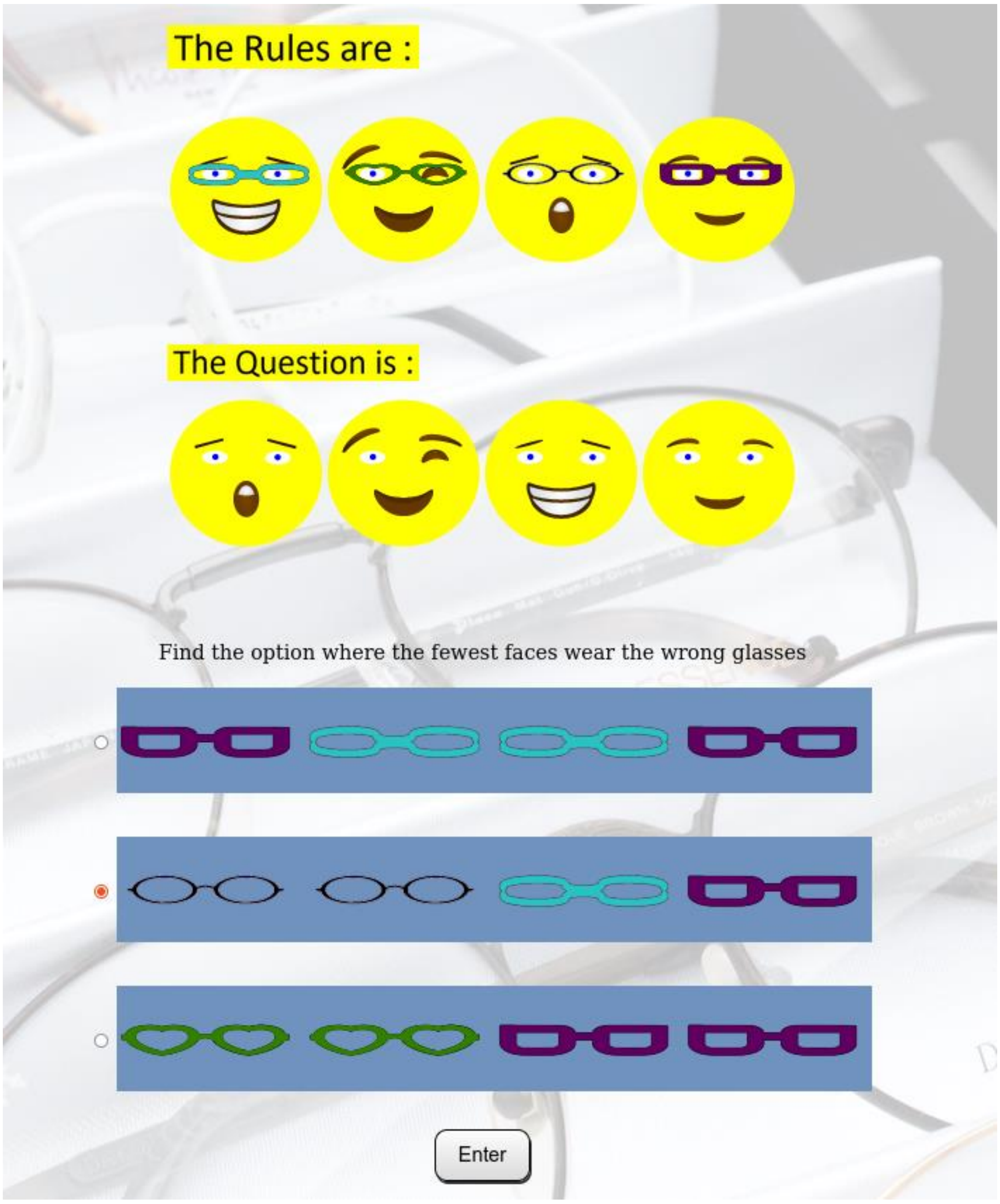
Your answer: NWWWSSSEESSEENENNWN

Glasses – An Easy level



Find the option where the fewest faces wear the wrong glasses

Glasses – A Medium level

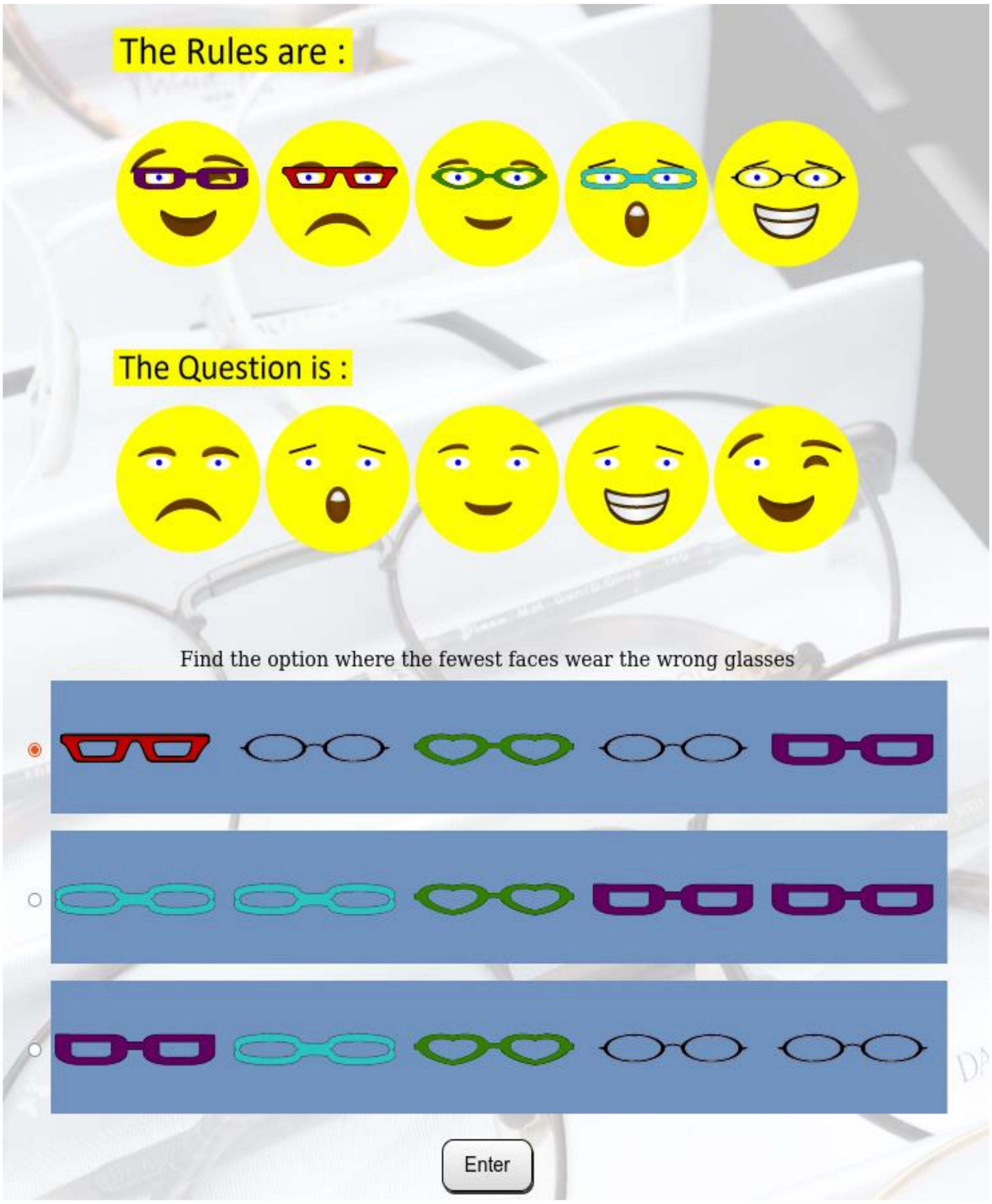


The Rules are :

The Question is :

Find the option where the fewest faces wear the wrong glasses

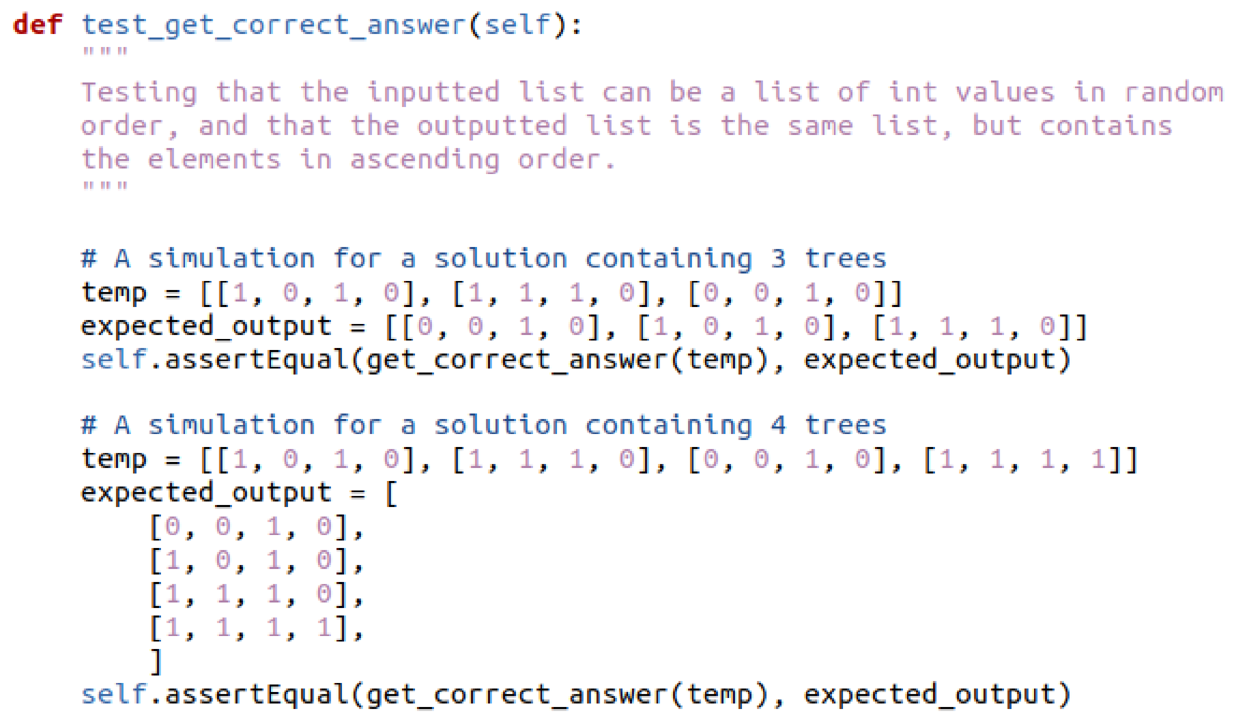
Glasses – A Hard level


The Rules are :
The Question is :
Find the option where the fewest faces wear the wrong glasses
Enter

Unit Test Results

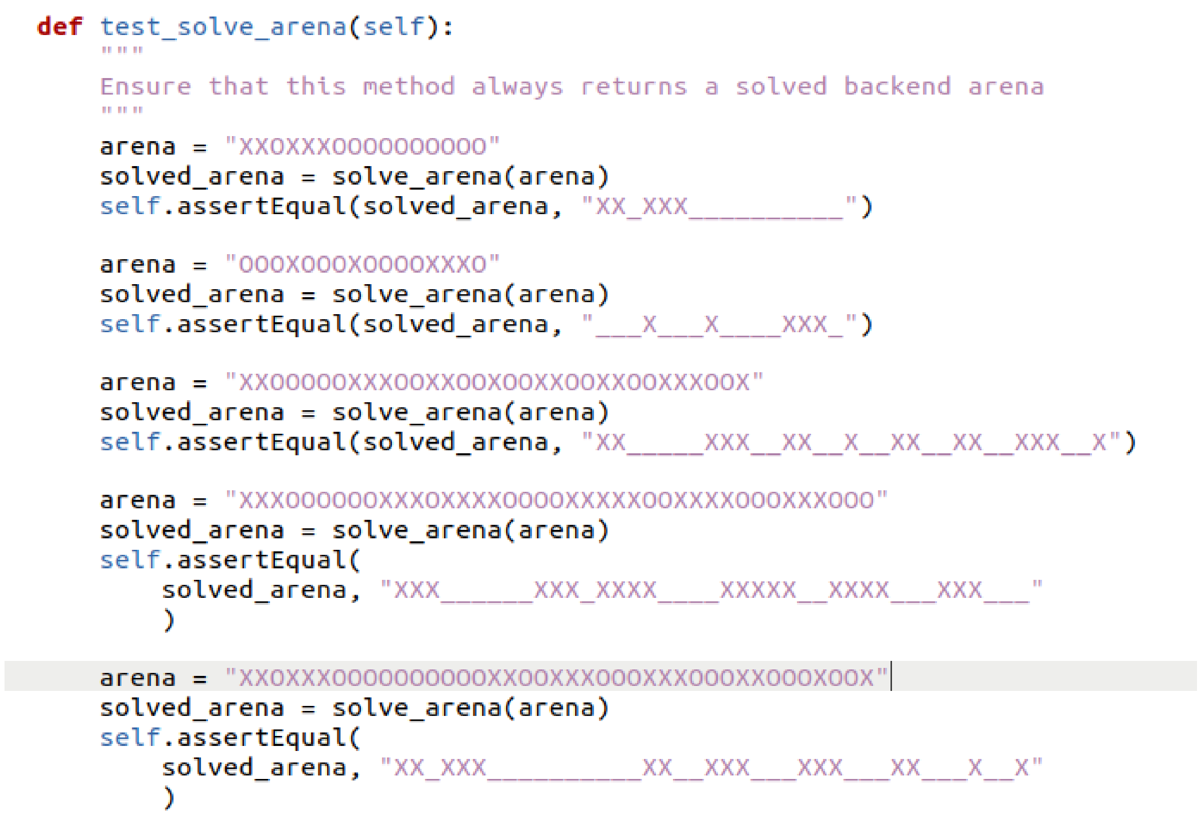
Snippets of code from each unit testing suite

Christmas Trees



```python
def test_get_correct_answer(self):
    """
    Testing that the inputted list can be a list of int values in random
    order, and that the outputted list is the same list, but contains
    the elements in ascending order.
    """

    # A simulation for a solution containing 3 trees
    temp = [[1, 0, 1, 0], [1, 1, 1, 0], [0, 0, 1, 0]]
    expected_output = [[0, 0, 1, 0], [1, 0, 1, 0], [1, 1, 1, 0]]
    self.assertEqual(get_correct_answer(temp), expected_output)

    # A simulation for a solution containing 4 trees
    temp = [[1, 0, 1, 0], [1, 1, 1, 0], [0, 0, 1, 0], [1, 1, 1, 1]]
    expected_output = [
        [0, 0, 1, 0],
        [1, 0, 1, 0],
        [1, 1, 1, 0],
        [1, 1, 1, 1],
        ]
    self.assertEqual(get_correct_answer(temp), expected_output)
```

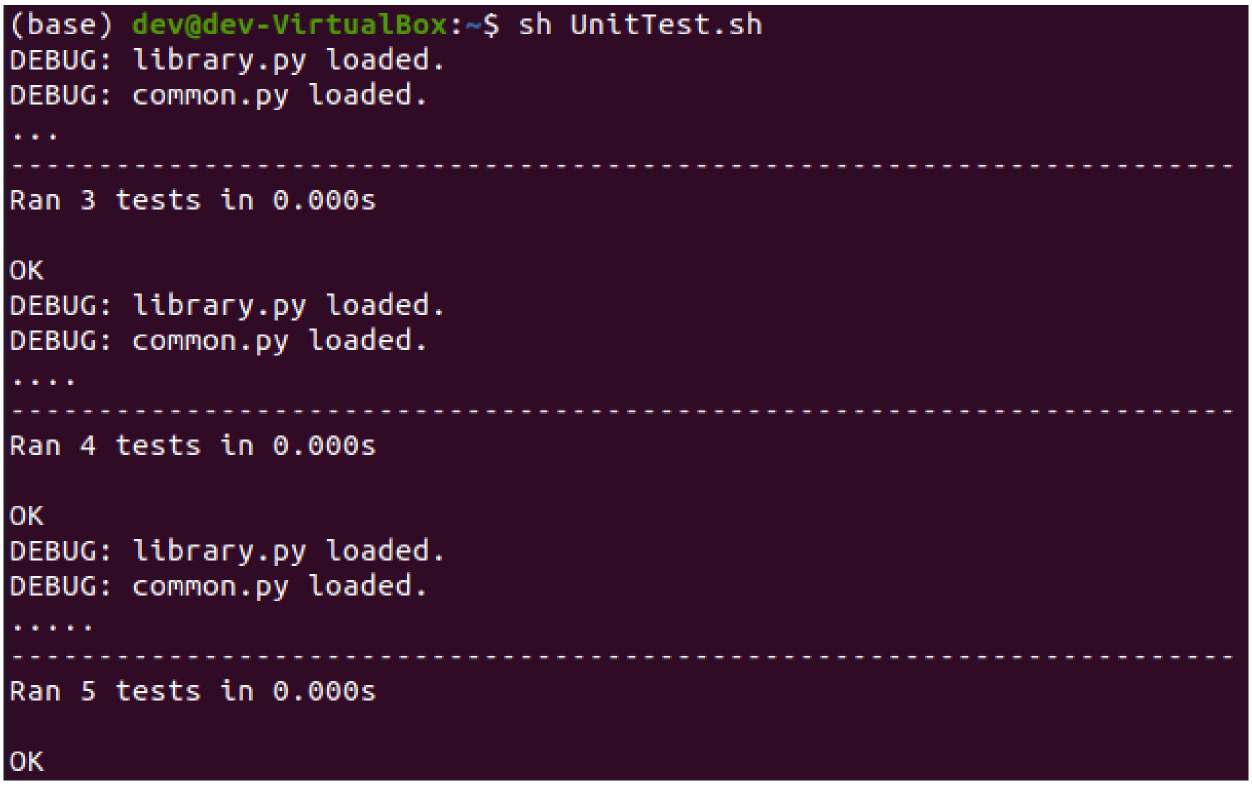
Clearing Snow



```python
def test_solve_arena(self):
    """
    Ensure that this method always returns a solved backend arena
    """
    arena = "XXOXXXOOOOOOOOOO"
    solved_arena = solve_arena(arena)
    self.assertEqual(solved_arena, "XX_XXX__________")

    arena = "OOOXOOOXOOOOXXXO"
    solved_arena = solve_arena(arena)
    self.assertEqual(solved_arena, "___X___X____XXX_")

    arena = "XXOOOOOXXXOOXXOOXOOXXOOXXOOXXXOOX"
    solved_arena = solve_arena(arena)
    self.assertEqual(solved_arena, "XX_____XXX__XX__X__XX__XX__XXX__X")

    arena = "XXXOOOOOOXXXOXXXXOOOOXXXXXOOXXXXOOOXXXOOO"
    solved_arena = solve_arena(arena)
    self.assertEqual(
        solved_arena, "XXX______XXX_XXXX____XXXXX__XXXX___XXX___"
        )

    arena = "XXOXXXOOOOOOOOOOXXOOXXXOOOXXXOOOXXOOOXOOX"
    solved_arena = solve_arena(arena)
    self.assertEqual(
        solved_arena, "XX_XXX__________XX__XXX___XXX___XX___X__X"
        )
```

Glasses

Result from running all Unit Tests concurrently using a shell script



```
(base) dev@dev-VirtualBox:~$ sh UnitTest.sh
DEBUG: library.py loaded.
DEBUG: common.py loaded.
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
DEBUG: library.py loaded.
DEBUG: common.py loaded.
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
DEBUG: library.py loaded.
DEBUG: common.py loaded.
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

User Evaluation Questionnaire

Please strongly agree, agree, disagree, or strongly disagree with the following statements:

Question	Christmas Trees	Clearing Snow	Glasses
The game has an intuitive user interface			
The game is appealing graphically			
The game has clear instructions			
The game is challenging			
The game scales in difficulty			
A Computational Thinking concept was required to win			
Specify a suitable age group for each game			

Please specify any issues you encountered while playing the games:

Additional Feedback / Comments on answers you chose:

Table 3: Questionnaire for User Evaluation

User Evaluation Raw Data

User 1

Question	Christmas Trees	Clearing Snow	Glasses
The game has an intuitive user interface	7	7	8
The game is appealing graphically	10	8	9
The game has clear instructions	7	4	9
The game is challenging	6	9	10
The game scales in difficulty	7	10	10
A Computational Thinking concept was required to win	6	10	7
Specify a suitable age group for each game	6-8	9-11	9-11

Please specify any issues you encountered while playing the games:

"I did not like the rules associated with Clearing Snow, I found them to be very ambiguous and did not get a good understanding of the game until after a few rounds."

Additional Feedback / Comments on answers you chose:

"The games were overall very colourful, and this made them enjoyable".

Table 4: Questionnaire for User 1

User 2

Question	Christmas Trees	Clearing Snow	Glasses
The game has an intuitive user interface	7	6	7
The game is appealing graphically	9	6	6
The game has clear instructions	8	5	7
The game is challenging	6	8	9
The game scales in difficulty	7	8	8
A Computational Thinking concept was required to win	8	8	8
Specify a suitable age group for each game	5-7	6-8	10-12

Please specify any issues you encountered while playing the games:

"For Christmas trees, when I chose the answer with one bauble on the first tree, I beat the level for the most part."

Additional Feedback / Comments on answers you chose:

"Clearing Snow was difficult to pick up at first."

Table 5: Questionnaire for User 2

User 3

Question	Christmas Trees	Clearing Snow	Glasses
The game has an intuitive user interface	8	6	8
The game is appealing graphically	8	7	8
The game has clear instructions	10	5	6
The game is challenging	4	7	9
The game scales in difficulty	8	9	8
A Computational Thinking concept was required to win	7	10	10
Specify a suitable age group for each game	4-6	7-9	10-12

Please specify any issues you encountered while playing the games:

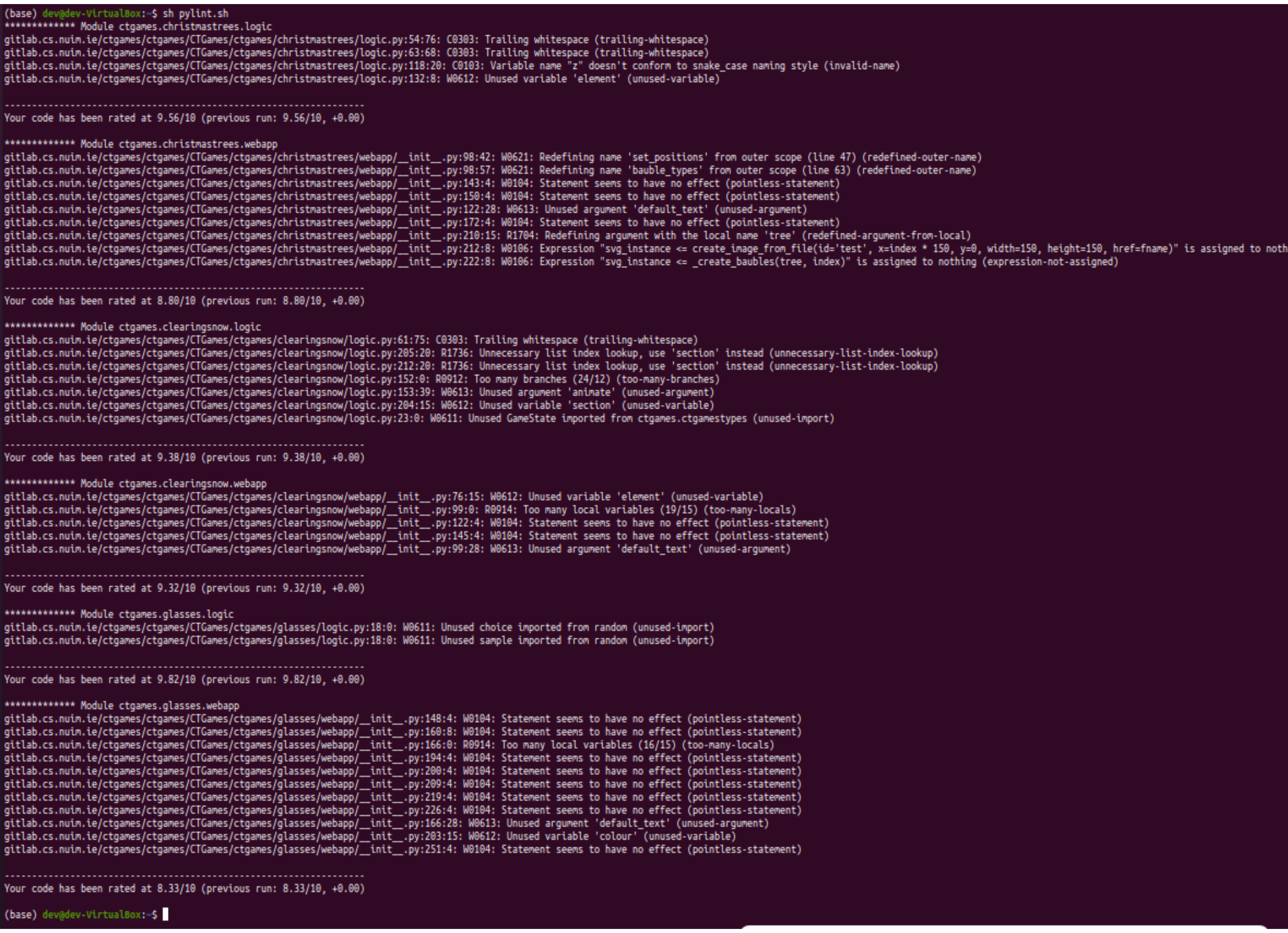
"I could not see where I had made the mistake in turning and moving the snow plough around the track".

Additional Feedback / Comments on answers you chose:

"The games got harder as I completed more levels, this kept it interesting. The colour is nice in Christmas Trees, and I enjoyed the seasonal theme in clearing snow."

Table 6: Questionnaire for User 3

Pylint Analysis



```
(base) dev@dev-VirtualBox:~$ sh pylint.sh
************* Module ctgames.christmastrees.logic
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/logic.py:54:76: C0303: Trailing whitespace (trailing-whitespace)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/logic.py:63:68: C0303: Trailing whitespace (trailing-whitespace)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/logic.py:118:20: C0103: Variable name "z" doesn't conform to snake_case naming style (invalid-name)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/logic.py:132:8: W0612: Unused variable 'element' (unused-variable)

------------------------------------------------------------------
Your code has been rated at 9.56/10 (previous run: 9.56/10, +0.00)

************* Module ctgames.christmastrees.webapp
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:98:42: W0621: Redefining name 'set_positions' from outer scope (line 47) (redefined-outer-name)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:98:57: W0621: Redefining name 'bauble_types' from outer scope (line 63) (redefined-outer-name)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:143:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:150:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:122:28: W0613: Unused argument 'default_text' (unused-argument)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:172:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:210:15: R1704: Redefining argument with the local name 'tree' (redefined-argument-from-local)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:212:8: W0106: Expression "svg_instance <= create_image_from_file(id='test', x=index * 150, y=0, width=150, height=150, href=fname)" is assigned to noth
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/christmastrees/webapp/__init__.py:222:8: W0106: Expression "svg_instance <= _create_baubles(tree, index)" is assigned to nothing (expression-not-assigned)

------------------------------------------------------------------
Your code has been rated at 8.80/10 (previous run: 8.80/10, +0.00)

************* Module ctgames.clearingsnow.logic
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:61:75: C0303: Trailing whitespace (trailing-whitespace)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:205:20: R1736: Unnecessary list index lookup, use 'section' instead (unnecessary-list-index-lookup)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:212:20: R1736: Unnecessary list index lookup, use 'section' instead (unnecessary-list-index-lookup)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:152:0: R0912: Too many branches (24/12) (too-many-branches)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:153:39: W0613: Unused argument 'animate' (unused-argument)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:204:15: W0612: Unused variable 'section' (unused-variable)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/logic.py:23:0: W0611: Unused GameState imported from ctgames.ctgamestypes (unused-import)

------------------------------------------------------------------
Your code has been rated at 9.38/10 (previous run: 9.38/10, +0.00)

************* Module ctgames.clearingsnow.webapp
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/webapp/__init__.py:76:15: W0612: Unused variable 'element' (unused-variable)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/webapp/__init__.py:99:0: R0914: Too many local variables (19/15) (too-many-locals)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/webapp/__init__.py:122:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/webapp/__init__.py:145:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/clearingsnow/webapp/__init__.py:99:28: W0613: Unused argument 'default_text' (unused-argument)

------------------------------------------------------------------
Your code has been rated at 9.32/10 (previous run: 9.32/10, +0.00)

************* Module ctgames.glasses.logic
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/logic.py:18:0: W0611: Unused choice imported from random (unused-import)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/logic.py:18:0: W0611: Unused sample imported from random (unused-import)

------------------------------------------------------------------
Your code has been rated at 9.82/10 (previous run: 9.82/10, +0.00)

************* Module ctgames.glasses.webapp
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:148:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:160:8: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:166:0: R0914: Too many local variables (16/15) (too-many-locals)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:194:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:200:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:209:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:219:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:226:4: W0104: Statement seems to have no effect (pointless-statement)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:166:28: W0613: Unused argument 'default_text' (unused-argument)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:203:15: W0612: Unused variable 'colour' (unused-variable)
gitlab.cs.nuim.ie/ctgames/ctgames/CTGames/ctgames/glasses/webapp/__init__.py:251:4: W0104: Statement seems to have no effect (pointless-statement)

------------------------------------------------------------------
Your code has been rated at 8.33/10 (previous run: 8.33/10, +0.00)

(base) dev@dev-VirtualBox:~$ 
```

Supporting Documentation

The code developed for this project, i.e., the three games and their unit testing files can be found in the attached ‘Supporting Documentation Upload’. They require the framework to be installed in set up in order to run, so have been uploaded for the purpose of reviewing the quality of Python code written. The set of Bebras tasks, developed games and their respective instructions are uploaded as screenshots in this document to give a demonstration of the code running, and are complemented by the demonstration video.