CSSE 1001
Friday 17 October
Assignment 3 – Design Document

Name: Joshua Hicks
Student Number: 41749450
Project Title: BBCUBER'S CUBE

1. Description

This project aims to create a program that will allow a Rubik's Cube to be simulated and solved on the computer. The simulator should be easy to control, and the controls should feel similar to a real cube. Replays and High scores should be saved for the next time the program is opened. There should be a variety of colour options, as people get used to their own colour schemes when solving. The modules that I used are time, tkinter, math, tkColorChooser, tkFileDialog, webbrowser and random. All of these are part of the standard library, so no extra modules will need to be downloaded.

2. User Interface

The Main layout is a dual screen application using Tkinter. The left hand screen will always show the speed solving cube, which the user can repeatedly scramble and solve. The right hand side can show a range of options. Below are four figures showing different configurations of this basic layout. Note how the right hand screen changes, whereas the left hand screen always shows a cube.

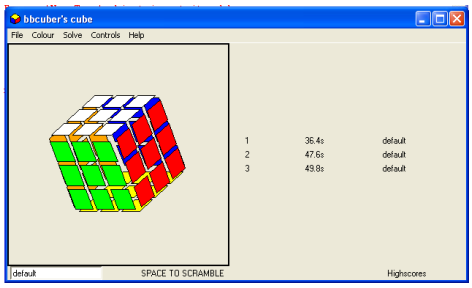

Figure 1

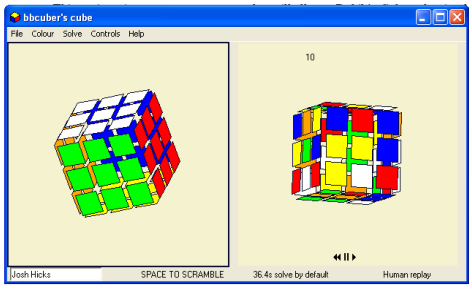

Figure 2

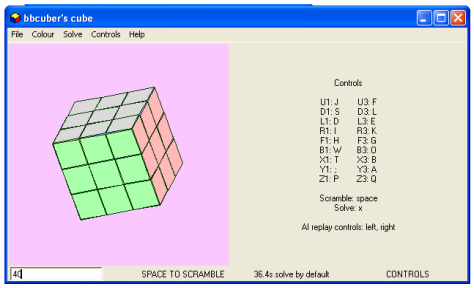

Figure 3

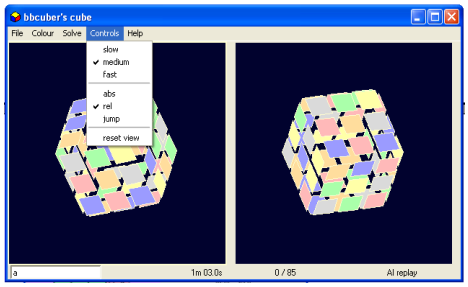

Figure 4

Figures 1 to 4 show a wide range of different options for the GUI. The left hand screen always remains the same. This contains a cube that can be manipulated using the keyboard. It also includes a text entry to enter user data as well as a timer which generally gives the time information for the current session. As shown in figures 1 – 4, the background colour, sticker colour and sticker size can be manipulated.

The right hand side of the screen holds two labels down the bottom for information as well as space that can hold either cubes or text. It can show high scores, an AI replay cube, a human replay cube or information about the game. When cubes are shown they have the same visual properties as the speed solving cube. There are five specific screens that are shown on the right hand side.

The main menu consists of:

- File for general program functions
- Colour for cube colour functions
- Solve for AI solving functions
- Controls for changing the axes rotation of the cubes
- Help for giving information about the program and the cube

The File menu consists of:

- Reset for unscrambling the cube
- Open for loading cube information from files
- Save for saving cube information to files
- Times for accessing the Highscore screen
- Exit for quitting the program

The Colour menu consists of:

- Config 1 for WRBYOG colourscheme
- Config 2 for WOBYRG colourscheme
- Config 3 for WGRBYO colourscheme
- Cusom for a custom colourscheme
- Customize to change the custom colourscheme
- Background to edit the background colour of the cubes

The Solve Menu consists of:

- Solve to access the AI replay screen and load solve information
- Pieces to change the solve option to the pieces option
- Petrus to solve with Petrus (unavailable)
- Heise to solve with Heise (unavailable)
- Fredrich to solve with Fredrich (unavailable)

The Controls menu consists of:

- Slow for a slow rotation speed of the axes
- Medium for a medium rotation speed of the axes
- Fast for a fast rotation speed of the axes
- Abs to rotate about the universal axes

- Rel to rotate about one of the cubes axes
- Jump to exchange the axes
- ResetView to return the axes of the simulator to the default position

The Help menu consists of:

- About to display information about the game
- Help to display control information
- See Petrus to open a link to the Lars Petrus Page
- See Heise to open a link to the Ryan Heise Page
- See Fredrich to open a link to the Jessica Fredrich Page

Other human/computer interaction occurs through:

- Name entry for user information and sticker size
- Highscore Page to open human replays
- Human Replay Page to control the human replays
- AI Replay Page to control the AI replay
- Simulator to control the speedsolving cube

The Name entry is at the bottom left hand corner and it's value is used for new highscores. If the value is a float and the user presses ctrl-z, then the sticker size will change. It is accessed by tabbing or by clicking on it. Enter or tab will move the focus on.

The Highscore Page shows the different records. If one is clicked, that record will be opened up and the human replay page will open in its place.

The Human Replay Page has there buttons. It has a reset, pause and play buttons. These control the Replay which has been loaded.

When the AI Replay Page is open, the user can control the orientation of the axes using the same controls as the simulator. The left and right arrow keys are used to run back and forth through the replay. For these controls to affect it, the canvas much be focused on. If the solver is done and the user presses the right arrow, the Highscore Page will take it's place.

The Simulator always appears on the left screen. When the focus is on it, various keys perform quarter turns, axes orientations, scrambling and AI solving.

3. Design

Summary

The data for a cube is stored in a Model. A model is a list of 20 initial position and current orientation pairs. The positions of these pairs within the array represent the different piece positions for a cube. In this way, all of the data for the cube is stored in only 40 numbers. While less could have been used (because you can deduce where the last pieces are), this is also quick and easy to manipulate. Note that no centre pieces are stored, because the positions are relative to the centres.

This data can then used by a Cube object. These contain 54 polygons representing the 54 different stickers on a cube. The data from a cube object can be converted to get the correct colours for the different polygons. Also, the Cube stores vectors representing the different cube axes, and using these can figure out which polygons to lower to show the cube in '3D'. Also, from these vectors, the program must find how the relative U,D,F etc faces match the absolute U,D,F faces when the cube has been rotated.

There is also a solver, which takes a model object and runs through all different combinations down to a set depth. This was done because if a move is repeated four times it is the same as not doing anything. By ensuring that at the end of every iteration, if a solution was not found, the cube would end up in the original position, it was easy to iterate through all possibilities. The advantage of this was that a single array could be manipulated and only one or two deep copies would have to be made no matter how deep the iteration.

The Dual screen works by continually grid and grid_forgetting widgets in a cell. There are four different widgets that get swapped in and out. Each of these is independent and all interactions are run through myApp. And because they are merely forgotten instead of erased, they can be updated and will keep their information until the next time they are accessed. The only one that has two functions is myApp.info, because it shows both control information and game information.

myApp also holds a Record, which inherits from list. This stores changes to the cube as well as the time such changes take place. It is updated when a turn occurs or when the speed solver cube is rotated. If a turn is given as an input to it's update function, it stores an array with time, a string 'turn' and a string for the given turn. If a move is given, it stores time, a string 'move' and the three unit vectors of the visual at that time. When the cube is solved this gets appended to the record list.

The replay display has information about the relative start time, the current instruction and the record it is displaying. Every time it runs an instruction, it calls itself back when the next instruction is due to occur. When the game is paused, it changes the relative start time, so that the cube appears to start in the same place when it keeps going. The reset just brings the cube back to the initial position and orientation by accessing the records initdata attribute and the first instruction, which is always a 'move'.

Classes:

The program consists of nine classes: MyApp, HighScore, Record, Model, PieceSolver, Cube, ReplayCube and ReplayData.

MyApp is the master class which holds everything else and runs most of the inputs. The only inputs that don't run directly thorough MyApp are the the cube axis rotations.

HighScore is a frame that shows the record replay information. It provides links to these different records when clicked

Record mainly does the time conversions from a float in seconds to hours, minutes and seconds. It has separate ways of dealing with negative times.

Model is the data structure that holds cube information in the form of initial position, current orientation pairs. The position in the list determines the current position.

PieceSolver is the class that solves a cube position using a piece by piece method. It saves solutions that it comes across, so while it now doesn't have to run recursive solvers, it has that abililty.

Cube is the visual cube canvas used for different cube applications. The orientation functions are bound directly to the Canvas. Sticker positions can be updated as data is given.

ReplayCube inherits from cube but differs because the only input are through the play, pause and reset buttons. It saves human replays to be rerun at a later stage.

ReplayData is a data holder for the different human replays. It stores, name, initial data, time and the list of moves and times needed for the replay.

The MyApp Class consists of:

- __init__(Tk): Tk is root. This constructs the MyApp. It sets up all the different instances of variables and gets the game ready to run

- keyinterp(event): event is a keyboard press. Depending on axis orientation it finds the correct Model function to call. It calls this and displays the change only if the cube is being solved.

- turn(function): function is a face turn for the simulator model. It turns, rerenders and check to see if it is complete

- replayunsolve(): undoes a move for the AI solver if possible
- replaysolve(): does the next move for the AI Solver if possible

- solve(): solve the current state using the selected method. If it’s not already solved then display the information in the right hand window

- updatereplayinfo(): change the text in the repinfo Label to show how far along the AI solve it is

- showhumanreplay(): brings up the human replay screen and changes the repcontrol text

- showhighscore(): brings up the highscore screen and changes the right labels accordingly.

- scramble(): restart the simulator with a random model. Display this and save the location into the replay data.

- reset(): resets the axis orientation of the simulator and displays it

- finished(): saves the new paths and resaves the records and then quits

- changecolour(): changes the colours of all cubes according to the settings

- pickcolour(): brings up a series of pick colour screens. The colour of the cubes is updated afterwards.

- pickbg(): brings up a pick colour screen and then updates the cubes accordingly

- render(Cube, list): chose a Cube to render with some Model data

- changespeed(): changes the axis rotation options of the AI replay cube and the simulator. If they are turning it stops them.

- timerstuff(float): float in seconds gets displayed in the timer in a nicer format and calls itself to run again if the message wasn’t predefined

- updaterecord(str): given a turn string if a quarter turn occurred. If it did store that data, otherwise store the current axis information

- loadreplay(int): int representing the replay number to load. The function loads it and shows the Human Replay Screen with appropriate label information

- about(): brings up the game information screen. Just gives basic information about who made the game and the easter egg

- helpme(): brings up the control information. Lists the different turns for the simulator.

- forgetme(widget): There are four different widgets that share the right hand screen. This function makes the others lose their grid geometry and grids this one to ensure that there are no packing problems.

- savecube(): if it's not already solved then you can save the current information

- opencube(): you can open a previous session.

- Seeheise(): brings up the Ryan Heise Page

- Seepetrus(): brings up the Lars Petrus Page

- Seefred(): brings up the Jessica Fredrich Page

- Setss(): called when ctrl-z is pressed in the name box. If it's a float then set the cube sticker size to that value.

The HighScore class consists of:

- __init__(Frame, function, ReplayCube): A Frame as it's parent, a function to call when the items get clicked and a ReplayCube to get it's values from.

- update(): updates information from it's ReplayCube. This ensures that the links will match the records. It is also more efficient than storing time twice.

- click(event): event is a mouse click. The function given in the constructor will be called and the widget from the event will determine the arguments.

The Record class consists of:

- __init__(str, float): corresponding to a name, time pair. The records only need to store this information.

- __lt__(Record)>bool: is used to sort a list of records. Returns true if less than.

- getTime()>str: returns the value in a nice string format. xh xm xs and the higher terms don't show if they're not needed.

The Model class consists of:

- __init__(): creates an empty list for the model data

- getData()>list: returns the data attribute as a list

- reset(): changes the data to that of a solved cube

- scramble(): iterates through 100 random face turns

- U1 through to R3: long and written out. In previous versions all this was written out in 4 lines, but this ran much faster and as it is the centre of the recursive solving method, I did it the long way.

The PieceSolver class inherits from Model and consists of:

- solve(int, list, list, str)>str: given a depth, the list of edges to be placed correctly, the list of corners to be placed correctly and any turns that should not be attempted because they would be redundant. If possible it finds a path to place the corners correctly and returns it as a string

- minsolve(int, list, list)>str: given a depth and the list of edges and corners, if possible this function will return a minimal path as a string

- solveCorner()>str: returns the path as a string. There is data for all the different stages and different function that are called for these different stages.

- solve2(int, list, str)>str: given a depth and a list of pieces to place correctly as well as an optional face to discard in the solving process. If possible returns a path to place the pieces correctly as a string

- minsolve2(int, list)>str: given a depth and a list of pieces to place. If possible to place them it will place them in a minimal path as a string

- solve3(int, list, list, str)> str: given a depth, a list of pieces to position correctly and a list of pieces to orient correctly as well as an optional argument to not count a face. Returns a path as a string

- minsolve3(int, list, list): given a depth, a list of pieces to position correctly and a list of pieces to orient correctly, this will find a minimal path and return it as a string if one exists.

- readfilestep(int): integer represents a stage in the first two levels. The function will load a file with the solved algorithms for that stage.

- readfilestep2(int): integer represents a stage in the top level. The function will load a file with the solved algorithms for that stage.

- writefilestep(int): integer represents the stage. The function will save the algorithms that have been solved.

- setup(): will load in the files and setup the controls.

- runalg(str): string represents a list of face turns. Each turn is read and performed individually.

The Cube class inherits from Canvas and consists of:

- __init__(tk, function): tk is the parent widget of the Cube. The function is an optional function to call when the axes change.

- resetview(): the axes are reset for the cube

- tmove(): the coordinates of the polygons representing stickers are updated.

- tcolours(list): the list is a list of sticker colours which is used to update the colours of the polygons representing stickers.

- linkrotate(event): event is a keyboard event. The event is handled to setup the resulting axis rotation.

- repeatrotate(): repeatedly calls rotate until the cube stops rotating

- rotate(): changes the axis positions depending on various settings.

- changeangle(int, str): int represents the angle that the cube will turn through. Str represents the type of rotation. Together they make the different rotation options.

- tadd3(list, list, list, list)>list: add 4 2D vectors and return a 2D vector as a list.

- tadd2(list, list)>list: add 2 2D vectors and return a 2D vector as a list

- tadd8(list, list)>list: take a 2D vector representing a translation and a 2x4 vector representing the coordinates of a sticker and return a 2x4 vector as a list that represents the corners of an adjacent sticker

- tadd23d(list, list)>list: add two 3D vectors and return the 3D resultant as a list

- trev(list)>list: reverse all values in the list. Represents a negative vector.

- tdiv(list, float)>list: divide a vector by a scalar and return a vector

- tmult(list, float)>list: multiply a vector by a scalar and return a vector

- tface(list, list)>list take in two vectors representing two sides of a face. From this return a a list representing all coordinates of all stickers on that faee.

- tconvertdata(list)>list: take a list of model data as piece position and return it as a list with colours relative to absolute sticker references

- rote(list, int)>list: rotate an edge about some number of times and return that edge as a list

- rotc(list, int)>list: rotate a corner about some number of times and return that corner as a lsit

- tcomp(list, int)>list: take a list of vectors and just return a list only holding the components along one axis.

- tlinecolour(str): change the colours of the polygons that represent stickers to be the str given. This will be the opposite of the background colour.

- Stopspinning(event): optional FocusOut event because it sometimes run when the widget is tabbed out of. It stops the cube spinning.

The ReplayCube class inherits from Cube and consists of:

- __init__(tk): constructor for the Replay Cube. It unbinds the rotation by keys and adds the control bar

- updatetimer(): updates the timer value to match the current time of the solve

- openrecords(): loads the records from files

- saverecords(): saves the records to files

- funcfd(): when the fd button is hit set the cube up to play

- runfd(): run the current instruction and callback in time for the next

- funcps(): pause the game ready to be restarted from the same position

- funcbkbk(): resets the replay to the initial position

- stoptimer(): stops the timer if it exists

- clickhandle(event): event is a mouseclick event on one of the controls. This function redirects the call to one of the other function.

- loadrecord(int): loads a record into the replay view and resets the cube to the start of the recording. Int represents the record number.

- returndata()>list: returns a list of name, time pairs. Used to load the highscore widget.

The ReplayData class inherits from list and consists of:

- __init__(str, list, float, *args): where str is the name of the person, list represents the initial cube data, float is the total time and *args is the actual replay data.

- __lt__(ReplayData)>bool: returns true if it is less than another. Used for sorting lists of replaydatas.

- __repr__()>str: gets rid of the many decimal places after floats for less memory usage in storing. Str represents the data in the record.