Choose Your Own Adventure Story

Jillian O’Connell

Department of Computer Science, Saint Joseph’s University

CSC 495: Computer Science Senior Project

Dr. Wei Chang

May 6, 2024

Contents

Choose Your Own Adventure Story 3
The Project 3
    The story 3
    Creating the scenes 4
    Creating the program for displaying the story 5
    Scene calibration 7
    Creating the program for detecting decisions 8
    Keypresses 9
    Challenges 10
Conclusions and Summary 11

Choose Your Own Adventure Story

This project was designed as a continuation of the Honors Capstone project that was started in the fall semester. In the fall semester, a first person animatic was created using the Unreal Engine. In the animatic, the viewer began in a room and went through a portal to a fantasy world. After walking through and exploring the world, the viewer exited through a different door. The project completed in the spring semester was designed as a continuation of the story from the last project, but with added user involvement.

The Project

The story

To begin, the story of the project was crafted. Because the story was intended to be a continuation of the story from the fall semester, it begins with the user deciding between two doors.

If the user chose door A, they would be brought to a scene with different kinds of flowers and would have had to choose between the violet or the orange flower. If the user chose the violet flower, they would finish the story with a bookshelf inspired by the summer season. If the user chose the orange flower, they would finish the story with a bookshelf inspired by the fall season.

If the user had chosen door B, they would be brought to a scene with a music box. Aat the end of the transition video, the user would be presented with two keys. If the left, blue key was chosen, the user would finish the story with a bookshelf inspired by the winter season. If the user chose the right, green key, they would finish the story with a bookshelf inspired by the spring season.

Creating the scenes

To implement this story, each scene was created using Maya. Maya is a 3d modeling software that allows the user to create 3d objects from scratch completed with animations. There are two different kinds of objects that one can begin creating a model with: Polygon Primitive and NURBS Primitive. A polygon primitive object has a series of vertices, edges, and faces where the faces are rectangular with hard edges. A NURBS primitive object has a series of vertices, edges, and faces, but the faces are curved surfaces with curved edges. Both types of objects have their advantages and are useful depending on the object one is trying to sculpt.

The start scene was created using cubic and spherical polygon primitive objects. Each object can be assigned a material, which controls the look of the light hitting the object, and a color. The flowers for scene 1A all began as 2D rectangles, and the stems began as cylinders. Each flower petal was sculpted starting from the 2D rectangles, and each was resized and moved to create the full flower. For scene 1B, the music box was created from a cube polygon primitive object. Each music note along the inside lid of the box was created using both cubic and spherical polygon primitive objects. The record base that the star sits on was created using NURBS circles. Finally, the star itself was created from a 2D rectangle; it was shaped to be a star before being extruded to become a 3D object. Each scene from level 2 of the story, all of the bookshelves and all items on them, were created using a series of objects. Many objects in the scene are a collection of four or more models sculpted and combined to create the full object that the user sees. For example, the snowman in scene 2C was made using spherical polygon primitives, cylindrical polygon primitives, and a cone polygon primitive for the nose.

Once the scene was modeled, animations could be added to create the transition videos from the start scene to scene 1A and scene 1B. Each animation was created using Maya’s “set driven key" and "set key" features. With animations and modeling complete, basic lighting had to be added into the scene so that the models would be visible when rendered. For each scene, directional lighting was used, and directional lighting simulates the sun. There were other lighting options available, but directional lighting was the most applicable because it was the most natural. The direction that the light was coming from and the intensity needed to be different for each scene based on the colors and locations of different objects in the scene.

Each scene could then be rendered out of Maya. Each time, the render settings needed to be updated including the file name, file type, blur factor, and frames to be rendered. When rendering, Maya exports the scene as a series of images. This causes animations to be distributed amongst images of each frame instead of being a cohesive video. Because of this, the images had to be imported into video editing software and cut together. Once that video was created, it could be exported from the editing software and added to the folder with all of the rendered images.

Creating the program for displaying the story

The C++ program was the glue of the project. Open CV was a tool used along with C++ to display the images and videos that were created in Maya. The first step to creating this program was linking Open CV with Visual Studio C++. If it was not linked perfectly, the open CV functions would not work correctly, and debugging would not be possible.

For this project, the laptop screen and a monitor were used. The monitor was used to show the window of what the webcam was reading as well as the output. Conversely, the laptop screen was used to display images and videos and was the main window for the user to view the story. The settings had to be created with these two displays in mind, and both windows were defined in the program.

The next step was displaying the scenes correctly. Open CV has a function called “imShow()” which is used to display images to the specified window. In this case, the images were displayed to the laptop window using this function. In order to continuously display these images to allow the user time to make their decision, the images had to be loaded in the main loop. If the image was loaded outside of the main loop or in a separate method, then they would be loaded for a second and then taken off of the screen. Additionally, the videos had to have the capability to be displayed. Each video is displayed as a series of image frames. Each image frame from the video was displayed the same way as the set images. The difference here was that the video frame being displayed changed with each iteration of the loop whereas displaying the image consisted of displaying the same image continuously.

In order to store the scenes and videos, there were two main classes that were used. The first was the Scene class that stores the scene name, file name of the scene, and a list of targets. The scene name must be unique in order to identify the scene that the user is currently at. The file name of the scene is used when displaying the image; it is used to get the image from the computer before displaying it to the laptop window. The list of targets is created from a series of target objects which are instances of the target class.

The target class includes the x and y coordinates of the location of the decision in a given scene as well as the RGB value at that set of coordinates. There is also the name of the target, which is arbitrary, and the video to be played if that decision is selected. This video must be the file name of the video found using the file path for videos defined at the beginning of the program. Finally, there is the name of the next scene which must match the name of a different scene object. This way, when a particular target is chosen, the program knows what video to play and what scene to go to next using different data fields with previously set values.

These classes can be used to create objects for each scene and decision. This is done in the init_scenes method which is called at the beginning of the program. Within this method, every scene that is a part of the story has an object defined for it. Every decision that is possible in the story is created through target objects and pushed into a list. That list is what is used for the list of targets in the scene objects. For any given decision, there can be multiple target objects created. This allows multiple x and y coordinates to be defined for each decision so that the program is efficient no matter the direction that the user is coming from when making a decision.

Scene calibration

Once the program had the capability to display images, display videos, and store all of the necessary information for each scene of the story, the code for calibration had to be created. This project used an external webcam that pointed at the laptop screen to watch for the user's decision. In order for the program to understand what the webcam was reading, it needed to use an open CV object called a "VideoCapture" to read from the port that the webcam was attached to. Each computer is slightly different with what port it uses when utilizing an external webcam. This lead to the creation of the list_ports() method. This method reads which ports of the computer are working, which ones are reading, and which ones are not working. This method only needed to be run once when implementing the program on a new computer. Typically, at least two ports would be identified as working: the built-in webcam and the external webcam. Once the programmer understands which port is applicable to the external camera, the CAM variable was set to that port to be used for reading from the webcam.

This new added ability to read from the external webcam could be used for calibrating each scene. To calibrate, the test_scenes() method had to be run. This method would display each scene one at a time and take a screen capture of what the external webcam saw each time. These images could then be imported into image reading software. For this project, the image software "IrfanView" was used. With the image imported, the mouse could be used to get the x and y coordinates as well as the RGB value at different points on the decision objects. These values would be recorded and then implemented into the program in the init_scenes() method. Each set of coordinates and color values became a new target value. Those target values were then added to a list of targets that was added into the scene objects for each scene.

Creating the program for detecting decisions

The main method for detecting decisions was the check_for_decision() method. The goal of this method was to look at the RGB color values at the x and y coordinates of the targets in each scene. The color value detected from the external webcam in each iteration of the loop would be compared to the color value originally assigned to the target object at that x and y pair of coordinates. The method for detecting decisions was responsible for doing this comparison.

When comparing the color values, the R value interpreted from the camera when the method was called would be subtracted from the R value set in the target object. The same would then be done for the G and the B values separately. These differences were then added up and the absolute value was taken in case there were any negative values. This single value could be understood as the difference in colors from what is expected to what is currently being read in from the external webcam. This difference would then be compared to the threshold value. If the difference was found to be greater than the threshold value, then the name of the target at that x and y coordinate would be understood as the user's decision.

Setting the threshold value included some experimentation. The RGB values at each set of coordinates could not be expected to remain constant when running the program. This is because if the camera focus changes, the shadows move, the sunlight hits the screen at a slightly different angle etc, then the RGB values would change. When setting the threshold values, it could not be set too low or these external factors would cause the program to interpret a decision when the user did not actually make one. Additionally, the threshold value could not be set too high or the program would never understand when a decision was made by the user. Keeping all of this in mind and viewing the differences when running the program multiple times during testing, the threshold value was set at 300.

Once a decision was detected, if there was a video specified to be played in the target object of the decision location, then the video would be played. After the video ended, the current scene would be set to the next scene which is identified from the target object as well. The loop would then return to the beginning, and the image of the next scene would be displayed since the current scene had been updated. If a decision were not to be detected, the loop would return to the start and continue to display the same scene image until a decision is made.

Keypresses

Open CV has a built in function that could be used to interpret key presses from the laptop's keyboard. In this program, these key presses were taken advantage of in order to add extra functionality to the project. The first key press was the spacebar; when the spacebar was pressed, a screen capture of what the external webcam was seeing at that moment would be taken and saved to the computer. This feature was extremely useful for testing. The next keypress was the escape, 'q', 'e', and 'x' keys which, when pressed, would turn off a flag thus causing the program to terminate. Because this program was implemented using a main loop, this key press interpretation was necessary to terminate the program when the story was complete.

The ‘s’ key was used to start the program. When the program is first run, the start scene is displayed on the laptop screen. The program does not start interpreting the user’s decision until the program is officially started using the ‘s’ key. This gives the user control over when the program starts detecting decisions. The ‘t’ key was used to run the test_scenes() method which is imperative for calibrating the scenes. The ‘2’ key could be used to run the test_scenes2() method. This method would check the targets of each scene and find the difference similarly to the decision detection method. This method is useful for testing because it allows the programmer to test how the program is calculating the difference without moving forward through the story if the threshold is reached. Finally, the ‘v’ key is used to run the play_video() method. This key is also useful for testing because it tests how the program is reading in and displaying a video without having to run the full program.

Challenges

The main challenge of the project was the C++ program. In order for the user to make decisions in real time, the program needed to be designed in an efficient enough way to process the decision in a matter of seconds. Additionally, the program needed to have access to the next scene that should be displayed and whether or not a video should be played. Finally, identifying multiple coordinates for each target in each scene allowed variations in how the user could cover the target and let the program understand that as a decision without having to perfectly cover the center of the target object.

To overcome these challenges, all of the main processing code is completed in a main loop. This way, detecting the decision, playing the video, if applicable, and displaying the current scene’s image would all be done every second with every iteration of the loop. Additionally, the scene class was created to allow a single object to store all necessary information for the current scene and the next scene should a decision be made. Finally, the target class was created in order to created multiple objects for each decision. This allowed multiple coordinate pairs to be assigned to each decision before being added to the scene object.

Conclusions and Summary

Overall, this project was completed with all intended features included, and the completion of the project closely aligned with the timeline that was outlined before beginning. Through the use of Maya, each scene was able to be completely unique since it was all made from scratch. With the combination of the C++ program and Open CV, the decisions were able to be detected efficiently so that the user could enjoy the show in real time. Though there were some challenges with the program itself along the way, it allowed different data structures and classes to be used to accomplish efficiency and accuracy. The goal of this project was to be a small scale version of an interactive show that could be implemented in a theme park, and that is exactly what was accomplished.