ESSAY NM2 - Interactive Visualization

Jan Kolkmeier

THE COURSE

Interactive Visualization was about making an “Interactive Visualization” for a shareholder. That is essentially a virtual (as in “3D”) simulation of the shareholders project, allowing interaction through the user to enrich the experience.
The work on our projects was accompanied by lectures about the matter and encouragements to experiment with Unity3D, our favoured tool to create 3D applications.
My group (Herjan Treurniet, Douwe-Bart Mulder and me) came in contact with our shareholder, Bert-Jan van Bijnum from the Telemedicine department. Bert-Jan asked us to work with the “scenarios” that were written for the U-Care project. U-Care is a service layer for integrated homecare systems. A central part is the front end for their product.
It is a user interface called Julie. It will be displayed and interacted with on any screen (televisions, computers, smartphones, ...) owned by both caretakers and -givers. It is used for communication, organization, entertainment and supervision. The mentioned “scenarios” are written fictional stories describing the interaction with Julie from the perspective of the user. They have an important position in the evaluation process of the innovations/developments made inside the U-Care project.
The problem with these scenarios is that they are linear and non-visual, limiting the insight to the requirements and causing different interpretations between colleagues. With the presumption was that it is easier to communicate over visual than over textual material, we decided that the task was to transform one of these scenarios to an “Interactive Visualization” that can be used to “play” through the scenario.
Said and done, at the end of the course we were able to deliver a completely playable scenario.

MY PART

As main developer I mostly worked on the Unity3D editor. After a great introduction to Unity3D held by Siewart van Wingerden I started experimenting with it.

The Level

I quickly felt confident in the editor and when we decided that we need to start with the building of the “level” (a part of a retirement home). I made it using the basic primitives Unity3D provides (“cube”, “sphere”, “ellipsoid” and “plane”). The result was more than satisfying and we kept it instead of going the “traditional” way of game development, designing the level in an external 3D-modeling tool, which would cost more time.

Character Control

The next step for me was to build a bird-perspective character-controller that can be navigated through the level by point and click. This brought two problems: We did not have a model for the character and we would need a waypoint system with path finding. The first was first solved by using an ellipsoid as placeholder until I replaced it with a character borrowed from the “Character Animations” example project from the Unity3D website. The path finding was a bit more difficult to solve, but eventually I ended up using an existing implementation of the A* algorithm that finds paths from a graph. The graph was semi-automatically generated: The nodes (waypoints) where placed manually into the level, while the the edges between the waypoints are generated on the fly through another script using raycasting.

Story System

The next part I worked on was a story-system that would allow us to author the scenario. Since we planned the story line by using a graph, I decided to stick to this paradigm and wrote a script that would wrap functions from to be the nodes of our story graph. These functions include information about the “connections” to the other nodes in the graph. Also a dialog system was written along this. More about the making of a simple story system can be found in the last section of this essay.

Camera Director

To visually tell a story, no matter if it’s a drawing, a film or a game, the perspective is an essential stylistic device, so I hacked together a “Camera Director” script, that allowed me to control the camera positions and targets through the story system.

Julie

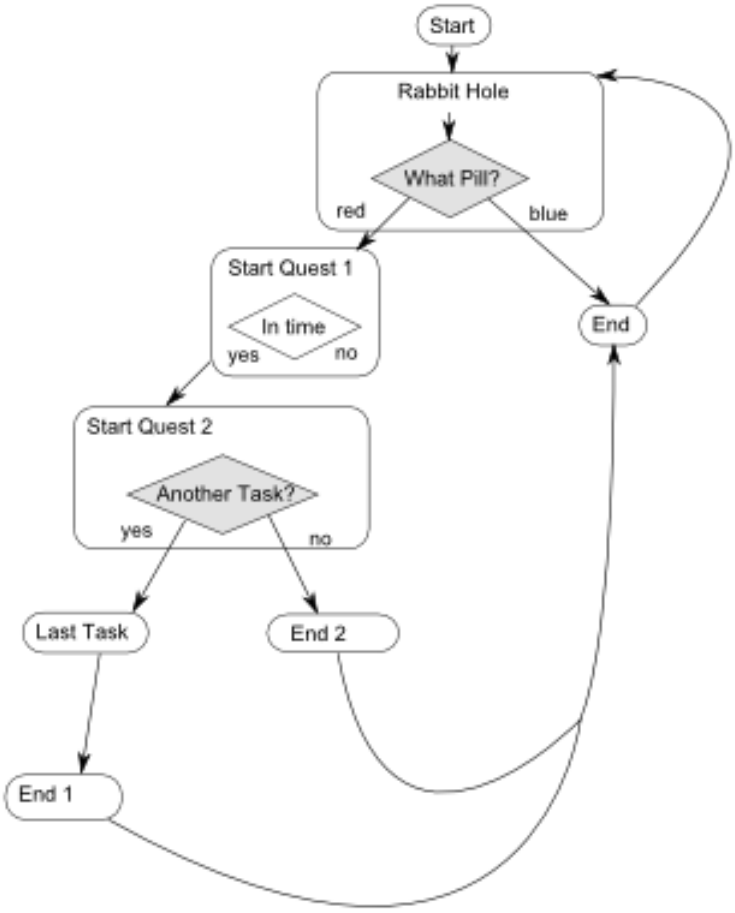
Since the interaction with Julie was an important element of the U-Care scenarios, put some extra effort to integrate it with the story system and the camera director (if you click on a Julie-screen, the camera moves from bird-perspective to a shoulder perspective etc.).

The Finnish

After implementing the scenario itself and polishing it up a bit I added a start and credit screen. This needs to be improved further with a help function and the likes, though. Also mentioned be the failed efforts to include the animations from the motion-capture session.

BUILDING A SIMPLE STORY SYSTEM FOR A GAME

The following tutorial will explain how a story and dialog system could be made in C# for Unity3D. Of course, this should be applicable to other game engines as well. At first lets take a look on how one could graphically represent a storyline:



The story starts with a "rabbit hole". In the example we will build it is a cube that will ask you to pick one of two pills as soon as you are close enough to talk with him. If the red one was picked, the player starts a quest. This quest needs to be completed in time before starting the second quest. It is noted whether he makes it in time or takes too long.
The second quest will eventually lead a decision, leading either directly to the end or after finishing a “last task”. Then one of the end sequences is displayed, depending on what decisions the player made.
The story can be played here [2]. Note that on the right side the story graph can be seen. A red marker is shown above the story node the player is currently in.
How can story graphs like the one above be turned into code? One possible way to do it is by transferring the nodes of the story to functions and wrapping them into a dialog system. At first, lets write out some of the nodes as functions. Note that this is pseudo-code and an implementation depends on the language you will use.

```
function RabbitHole() {
   while(Distance(Player, Cube)>1m) {
     wait
   }

   AskQuestion(
          Question = "What Pill?”,
```

```
            Answers  = [("Red Pill", StartQuest1),
                        ("Blue Pill", EarlyEnd)]
        )
}
```

This is the first node of the graph represented. We wait until the player is near the cube, then we will pop up a dialog question. The dialog requires two parameters - the phrase of the question, in this case “What Pill?”, and the possible answers. An answer consists of two elements: The text of the answer and the function of the next story node that should be called when this answer is given.

```
function StartQuest1() {
   while(!Confirmed(“Find the little cube!”)) {
     wait
   }

   time = 60s

   while(Distance(Player, LittleCube)>1m) {
      time = max(time-1, 0)
      ShowTimer(time)
      wait
   }

   if (time>0) {
      inTime = true
   } else {
      inTime = false
   }

   StartQuest2()
}
```

This function is a bit more elaborate. The “Confirmed” function is a dialog that just has an “Ok” button. The while loop will run until this button is clicked. Then we set up a timer, and wait until the player has completed the task, i.e. found the little cube. Then we set the variable “inTime” depending on whether the player did it in time. The next quest starts immediately.

```
function StartQuest2() {
   while(!Confirmed(“Find something to eat!”)) {
     wait
   }

   while(Distance(Player, Cake)>1) {
     wait
   }

   AskQuestion(
          Question = "What do?",
```

```
            Answers  = [("Eat it yourself", "End1"),
                        ("Bring to little cube", "End2")]
        )
}
```

Here is nothing new. Lets see how an “end” could look like.

```
function End1() {
   while(Player.EatCake()) {
     wait
   }

   if (inTime) {
     while(!Confirmed("You are fast, but a heartless Person!")){
       wait
     }
   } else {
     while(!Confirmed("You are slow and heartless!!")) {
       wait
     }
   }
   Quit()
}
```

With “Player.EatCake()” we would play an “eating” animation and wait for it to be finished before showing the result of the player depending on his history. “End2()” would look similar, but showing other results.
How can this be transferred to usable C# code to use in Unity3D? It’s not as easy as it seems, especially when it comes to the integration of the dialogs. The easiest way would be to wrap the whole thing into the “OnGUI” function. This however is not possible because one can not start coroutines from “OnGUI”, yielding tricks like using the while loops impossible. It would have to call the function of a story node once per frame instead of just once, which would require some extra logic.
Instead we split up GUI and Story logic by introducing the following classes:

- **Answer** class is used to create objects that contain two strings, the answer and the function that should be called when this answer is chosen. Just as introduced in the “RabbitHole()” function in the beginning.
- **Dialog** objects contain “Answer” objects and a “question” string.
- **Display** is the script that renders one Dialog object at a time, or nothing.
- **StoryPath** here all functions that represent the story nodes are included as coroutines along with some helper functions.

Lets first look at the StoryPath script. As said before we want every node-function to be started as a coroutine, so that we can spare some control logic and script the story more naturally. This would make the “RabbitHole” function look like this:

```
IEnummerator RabbitHole() {
    while(Vector3.Distance(player.transform.position,
              cube.transform.position)>1.0f) {
        yield return null;
```

```
        }

        display.SetDialog(
            new Dialog("Hello, what Pill?",
                Answer("Red Pill", "StartQuest1"),
                Answer("Blue Pill", "EarlyEnd")
            )
        );
    }
```

As you can see, the structure is pretty much the same as in the pseudo-code. “display” is an instance of the Display class. The function “SetDialog” simply activates the passed Dialog on the GUI. The constructor of the “Dialog” class is “Dialog(String question, Answer a1, Answer a2, Answer a3, ...)”. In this case the functions/story nodes we would move to by clicking one of the buttons are “StartQuest1” and “EarlyEnd”.
Lets have a look how this dialog object is displayed by the “Display” script and how it gives feedback to the StoryPath when a button is clicked:

```
public class Display : MonoBehaviour {
    StoryPath story;
    Dialog currentDialog;
    bool breaker = false;
    void Start() {
        story = transform.GetComponent<StoryPath>();
        Reset();
    }

   public void Reset() {
       SetDialog(new Dialog());
   }

   public void SetDialog(Dialog dia) {
       story.FreezePlayer(true);
       currentDialog = dia;
   }

    void OnGUI() {

                [...]

        if(currentDialog.answers.Length>0) {
            for(int i=0; i<currentDialog.answers.Length; i++) {
                 if(GUILayout.Button(currentDialog.answers[i].text)) {
                    story.stack = currentDialog.answers[i].function;
                    Reset();
                    story.FreezePlayer(false);
                    breaker = true;
                 }
             }
        }
```

```
                [...]

    }
```

In essence the trick is to write back the function name to a variable called “stack” if the corresponding button is pressed. Also the “currentDialog” object is replaced with an empty dialog by calling the “Reset” function which stops the Display class from rendering anything. The next example gets us back to the “StoryPath” class, to show how the function is called from the “stack”:

```
public class StoryPath : MonoBehaviour {
    Display display;
    public string stack;

    void Start() {
        stack = "";
        display = transform.GetComponent("Display") as Display;
        SendMessage("RabbitHole");
    }

    void Update() {
        if (stack != "") {
            SendMessage(stack);
            stack = "";
        }
    }
                [...]
}
```

Note two things: The name of the function we initially stored in the “Answer()” constructor is now read from the “stack” before getting executed by “SendMessage” once. This happens immediately after the button was pressed in the GUI, where the appropriate function name was written to the “stack”. Voilà, the dialogs are integrated to our story system.
So by clicking a button we traversed from one story node in the graph to another by making a decision. Of course these traversals can also be made without clicking through a dialog. Lets have a look at one last example to show how:

```
IEnumerator StartQuest1() {
    Destroy(GameObject.Find("RedPill"));

    yield return StartCoroutine(WaitForAnswer(
        new Dialog("Find the little cube!", Answer("Ok", ""))));

    float time = 15.0f;
    bool warning = false;
    while(Vector3.Distance(player.transform.position,
                    littleCube.transform.position)>7.0f) {
        display.SetTimer(time);
        time = Mathf.Max(time-Time.deltaTime, 0);
```

```
            yield return null;
        }

        display.SetTimer(-1.0f);
        inTime = time>0.0f;

        StartCoroutine("StartQuest2");
    }

    IEnumerator WaitForAnswer(Dialog d) {
        display.SetDialog(d);
        display.breaker = false;
        while(true) {
            if(!display.breaker)
                yield return null;
            else
                yield break;
        }
    }
```

This is the “StartQuest1” function translated from pseudo-code. Here we traverse to the “StartQuest2” story node by just starting it as next coroutine at the end of “StartQuest1”. Also a helper, “WaitForAnswer”, is introduced. It represents the “Confirmed” function from the pseudo-code. It can be started as coroutine. The dialog that got passed will be initiated and the coroutine won’t stop before a button is clicked. This helper is useful in this example because we want to start the timer just after the user read the instructions, not immediately. I am sure this can be written much nicer, but i don’t know all the tricks syntactic sugar of C# yet, so this has to do it for now. I worked out a simple example with some extra features in this project [1]. If you just want to see how it looks like, check out the webplayer build [2].

[1] http://ewi1544.ewi.utwente.nl/~jan/storytutorial/StoryTutorial.zip
[2] http://ewi1544.ewi.utwente.nl/~jan/storytutorial/index.html