FAKULTÄT FÜR
INFORMATIK

“A Projection and SensFloor based Augmented Reality Educational Game for Museums”

Department of Simulation and Graphics

DE Project Report

Asema Hassan

210492

Supervisor:

Jun.-Prof. Dr. Christian Hansen

23.02.2017

Table of Contents

Introduction

This game prototype is developed as part of Digital Engineering (DE) project. The project falls under the category of DE module description, which includes use of hardware and software together.

This report is intended to help future students to understand the development stages and code of the game prototype. Since, the project has gone through different versions and have been tackled by two different teams. It necessary to summarise the important features and the final version of the project in report.

In this report, I will discuss the major development stages of the game prototype and highlight the important main features with an overview of code documentation.

Overview

The purpose of project is to make an educational game for kids who visits museums. The project idea was developed after doing few surveys with the museum representatives and gathering requirements from them.

After finalising their specifications, one museum was finalised to work with. Which is Museum of Natural History in Magdeburg, their main idea about game prototype was to show life in the tundra.

The game is to be played as part of a guided tour through the museum. It will be played at the end of the tour not to test the pupils understanding of the given information, but to deepen the learning effect of the guided tour.

The pupils should be able to review and expand the information given in the guided tour while playing the game. Since the game lets the player act as a glacial man who is lost in tundra and trying to head back to his land, the pupils will be able to become immersed in the historical lifestyle.

Because of the positive experience while playing the game, the new knowledge will be easier to remember, leading to a better learning effect. The game can also be adapted to be used in different museums or for special exhibitions. It can also be greatly extended to cover other aspects of life in the Tundra or general ice age respectively. For learning effect there are some short questions that pupils have to answer while playing game, in order to remember every stage of Tundra life.

Game Design

Idea

A game with a unique user interface will let kids in a museum / educational environment play the role of an inuit hunter. The main goal of game is to learn about the tundra environment and complete level by reaching the camp again.

Story

The narrative is mostly environmental with some non-diegetic storytelling elements (questions depicting tundra life).

A hunter is separated from his trusty dogs when a wind gust hits them and his sleigh is turning over. He wakes up a few minutes later when one of his dogs returns to bring him back to camp. The sleigh is severely damaged and the other dogs are already back in camp. The hunter has to follow the dog through the tundra and forage some resources on their way.

The setting is the tundra of the northern hemisphere, characterized by meager vegetation, harsh weather conditions and permafrost, which makes agriculture impossible; set during an unspecified time period in the 20th century. The native people of these lands are foragers and hunters.

Figure 1: Illustrative examples of tundra life; indigenous people (inuit) and landscape.

User Interface

The player interacts with the game by stepping on a SensFloor. There are two displays: the floor itself which maps user input coordinates directly to screen space coordinates (like a touch screen) and a bigger front screen in front of player. The following modes are supported by the game:

(a) Directly selecting user interface elements by stepping on them on the floor.
(b) Walking on the spot to move the player character forwards.
(c) Walking on the spot but moving sideways to move the player character to the left or right from the center point.

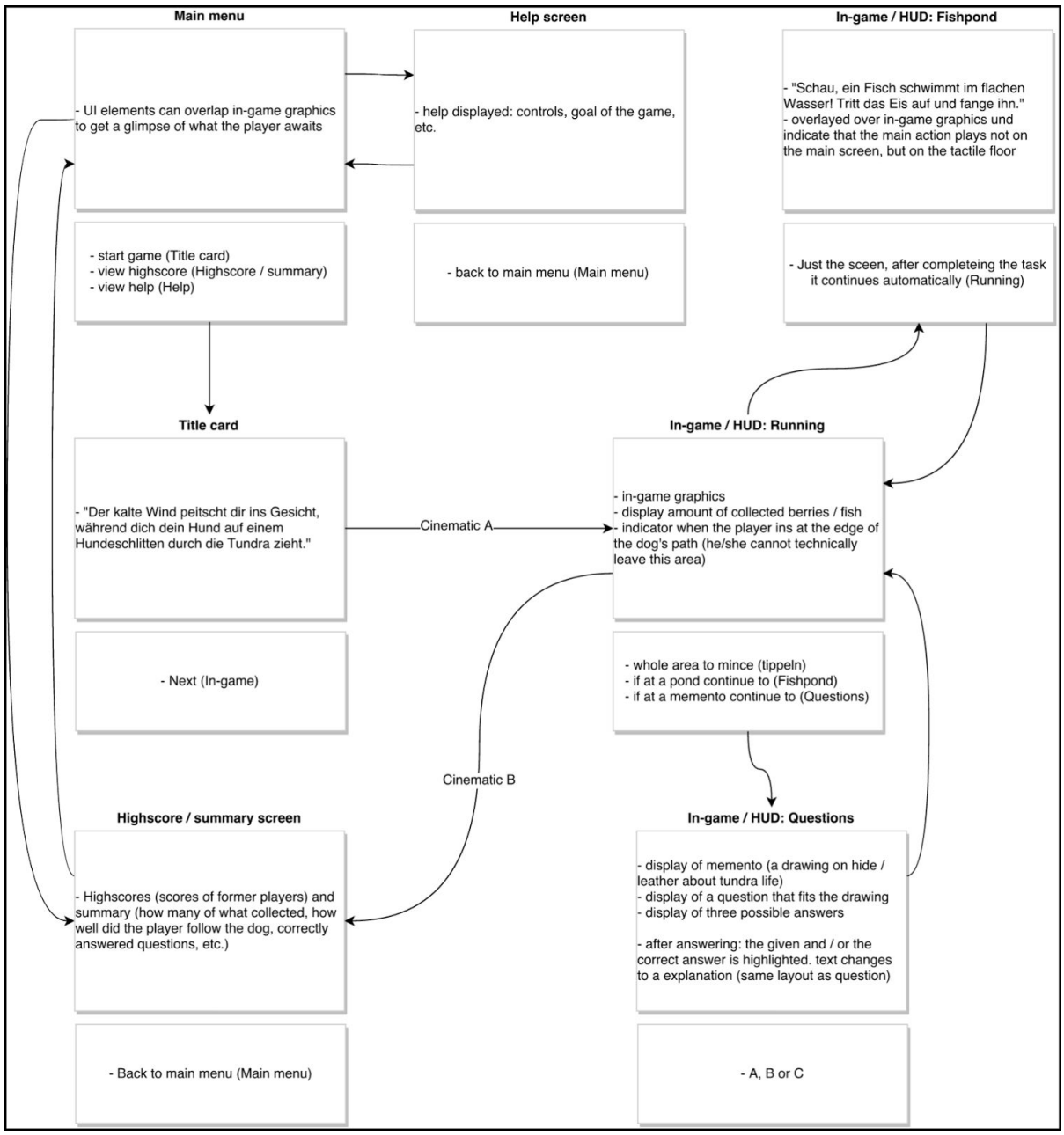


Figure 2: User Interface Design Flow Diagram

Gameplay

Goal

The player needs to reach the camp by (a) following the dog in a proper speed (not falling behind), (b) gaining as many score points as possible by collecting berries, cracking the icy surfaces of fish ponds and answering questions and (c) do all that without depleting his / her energy completely. See the following section regarding the gameplay aspects.

a) Following the dog

During core gameplay the player follows the dog (*in actual prototype a model of wolf is used*) on a predefined path (the “rail”) which is a curved line through the terrain. He does so by walking on the spot. On his way the player can sidetrack the main path to collect items and trigger gameplay events by walking on the spot but moving sideways. The player can thus influence the following parameters:

(a) The speed of the player character. The goal is to stay close behind the dog.
(b) The deviation from the rail, perpendicular to the main path. The goal is to collect gameplay objects.

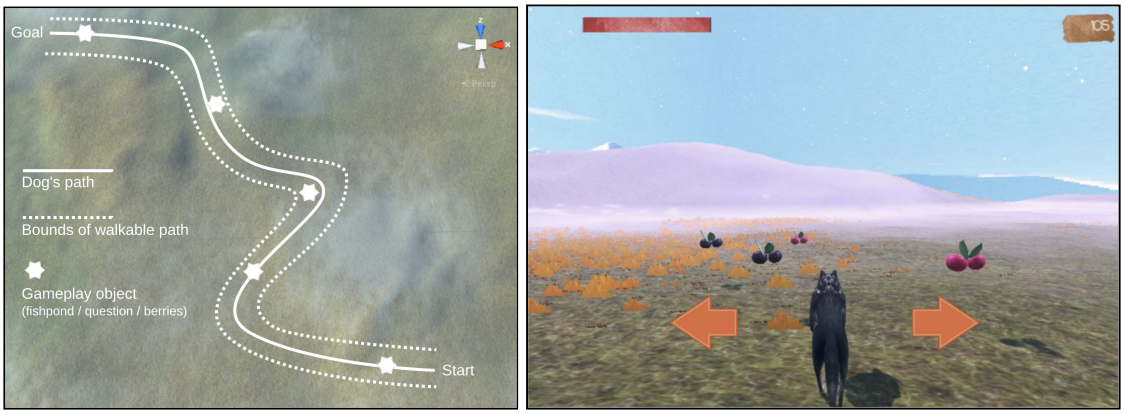


Figure 3: (a) Schematic illustration of gameplay mechanics and elements. (b) 3D Runner scene view player following dog

b) Gameplay Object: Items

Berries

The berries are represented by a 3D model in two types. Redberry depicting poison with -ve points and blackberry as healthy to gain +ve points. The berries are rotating a few inches above the ground.

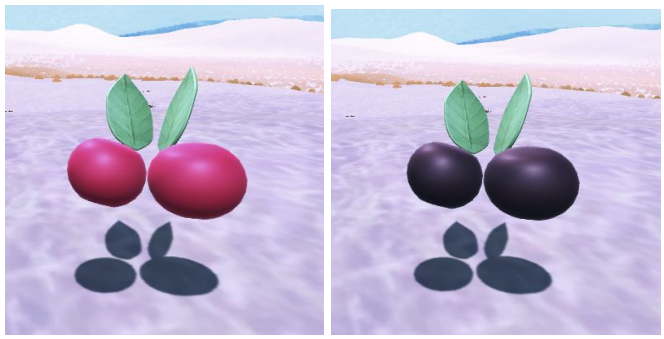


Figure 4: Berries model used in game prototype

Fish Pond

The fish pond trigger represented by a fish bucket is also hovering a few inches above the ground. When player collides with this bucket, it will be taken to a fish pond environment. The player will need to focus on the screen on the floor. This is indicated by a text phrase on the front screen. Stomping on the ice several times will break it. Then stepping on a fish will add points to player score and will return player back to the runner scene from same stage and time.

Figure 5: (a)Fish bucket to Trigger (b) Fishpond scene

Quiz

The quiz trigger represented by a question mark, rotating a few inches above the ground.The hunter will remember moments from his life. The front screen will prompt a scene from tundra life and a related question with the hint to answer on the floor. The floor screen shows four possible answers, arranged 2 by 2.

Figure 6: (a) 3D model used for trigger (b) Quiz questions regarding tundra (c) Possible options

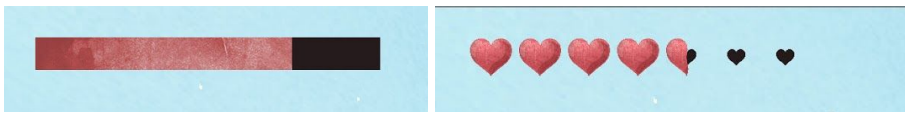
Selecting the correct answer adds 5 (due to higher challenge) to the score at the end, selecting the wrong one highlights the correct answer. The main screen shows additional information regarding the question. After player response has recorded, the game returns to the core gameplay of following the dog.

c) Avoid depleting energy (surviving)

The player's life energy sink because of the harsh temperatures. At zero the game will be over. The energy is based on time of the level and as time goes up the energy is decreasing so the player needs to survive within a limited time frame taking care of it health.

HUD

The HUD shows the current score in the top right corner and the energy in the top left corner. There are two options (one quick and one costly):



An energy bar quick option showing player health is implemented in prototype, the other possible option is to represent with hearts which could be that half of a heart shows 15% of energy (approx).

Start Screen

A title card is displayed reading "Der kalte Wind peitscht dir ins Gesicht, während dich deine Hunde auf einem Schlitten durch die Kälte ziehen." (The cold wind is beating your face as your dogs are pulling you through the tundra on a sleigh.)

Option 1

Another title card is displayed reading "Der unwegsame Grund lässt deinen Schlitten kippen. Ein treuer Freund kommt zurück, um dich zum Lager zu führen." (The pathless ground forces the sleigh to turn over. A loyal friend comes back to show you the way to your camp.)

Option 2

Same as Option 1 but the scene is actually played out. Needs model of sleigh, hunter and additional animation, too costly and time intensive for this prototype.

Ending

The camera slowly rotates around the campfire, in the background is the tent and next to the hunter's dog is lying next to it. The hunter's presence is only implied since the player just reach the campfire. The screen fades to the high score screen.

Sound and Music

The game features natural sounds for

(a) Wind and gusts of wind
(b) Footsteps on grass / snow
(c) Stomping on ice
(d) Eating berries
(e) Dog's barking and panting
(f) Camp fire crackling (for ending scene)
(g) Nuzzling hide up against hide (for interface feedback since the interface will mimic a leather / hide look)
(h) The drum sound of a Qilaut drum (flat drum without boiler) when points are added, think fanfare.

Music played on an Icelandic langspil featured during the main menu and end scene / highscore screen. While the langspil is not directly used by the indigenous people of the tundra, it is related to the fidla which is very similar to instruments actually used by them (probably introduced by seafaring norse).

Game Development

Framework

- The game is developed for the windows operating system Windows 7 or higher.
- The development environment used is the Unity 5.4.0 engine.
- The game 2D elements are created using Adobe Illustrator and Photoshop.
- The game 3D elements are created in Autodesk 3ds Max.
- The hardware setup for the game consists of two projectors and a 2m x 1m SensFloor.

Stages

The game prototype has passed through different stages of development. In the first stage of the prototype the game was designed to match the requirements of the life in the Tundra as described in the requirements document. Those requirements were requested by the Museum of Natural History in Magdeburg.

In the second stage the game design was finalized with the designers involved in the prototype development. As the prototype contains both 2D and 3D game designs, it was important to finalize the feature list for both scenes, with the limitation of design. Finally, after

the design mockups were created the level design was made in Unity with core functionality of gameplay with proper integration with SensFloor and multiple projectors.

In the third stage, the gameplay was improved with a better implementation of SensFloor data integration and new features were added to the game i.e Learning quiz, sound game for feedback testing.

SensFloor

In order to work with Sensfloor, in the first stage of development a Dynamic Link Library (DLL) was created written in C++ that reads data from floor through serial port “COM3”. The DLL was added into the Unity “**Assets/Plugins**” folder. The methods of the DLL were called in <FloorController_CPlusPlus.cs> but due to loading of DLL and waiting for the data stream from floor, there was a huge lags in response. Also once DLL is loaded into Unity, it was not unloaded. The C++ implementation used a blocking call that becomes unresponsive when there is no interaction with floor and hence, the Unity crashes.

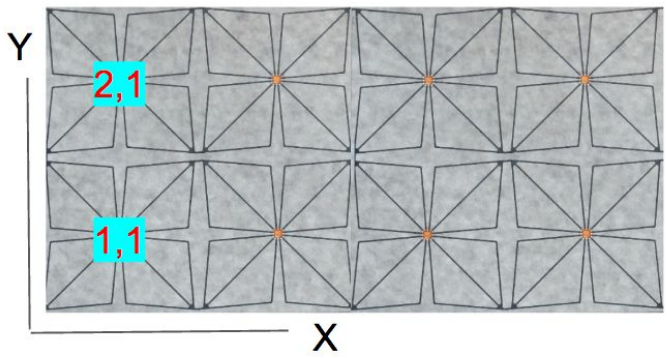


Figure 7: SensFloor 1m x 2m design

In order to improve the game response and quality of Sensfloor data, the serial communication was established from within Unity. As C# (.NET framework) supports it pretty well. A <FloorControllerCSharp.cs> class reads the data stream from the floor and refines it to be used in game.The main issue of crashing of Unity and loading/unloading DLL was completely solved. Although, the issue with stream of data and response time is still on stake. Hence, needs more improvement and evaluation.

The floor is by SensFloor, it is the major source of input in the game. It is designed and manufactured by the company FutureShape and has a size of 2m x 1m. The floor is able to sense the position of a certain cell and the weight of the user on the triangle that is under the user’s foot.

```
public void Run ()
{

    Action startReading = null;
        startReading = delegate
        {
            _serialPort.BaseStream.BeginRead(_serialBuffer, 0, _serialBuffer.Length,
                delegate (IAsyncResult ar)
                {
                    try
                    {
                        int actualLength = _serialPort.BaseStream.EndRead(ar);
                        byte[] received = new byte[actualLength];
                        Buffer.BlockCopy(_serialBuffer, 0, received, 0, actualLength);
                        if (isFloorMessageValid(received))
                        {
                            //Debug.Log(BitConverter.ToString(received));
                            FloorData floorData = new FloorData();

                            //Exchanged index values for x and y (as x is rows and y is cols)
                            floorData.x = BitConverter.ToInt32(new byte[] { received[3], 0, 0, 0 }, 0);
                            floorData.y = BitConverter.ToInt32(new byte[] { received[2], 0, 0, 0 }, 0);
                            floorData.t1 = BitConverter.ToInt32(new byte[] { received[8], 0, 0, 0 }, 0);
                            floorData.t2 = BitConverter.ToInt32(new byte[] { received[9], 0, 0, 0 }, 0);
                            floorData.t3 = BitConverter.ToInt32(new byte[] { received[10], 0, 0, 0 }, 0);
                            floorData.t4 = BitConverter.ToInt32(new byte[] { received[11], 0, 0, 0 }, 0);
                            floorData.t5 = BitConverter.ToInt32(new byte[] { received[12], 0, 0, 0 }, 0);
                            floorData.t6 = BitConverter.ToInt32(new byte[] { received[13], 0, 0, 0 }, 0);
                            floorData.t7 = BitConverter.ToInt32(new byte[] { received[14], 0, 0, 0 }, 0);
                            floorData.t8 = BitConverter.ToInt32(new byte[] { received[15], 0, 0, 0 }, 0);

                            PrintFloorDataToConsole(floorData);

                            _floorData = floorData;
                        }
                    }
                    catch (IOException e)
                    {
                        Debug.LogException(e);
                    }
                }, null);
            if (_stopThread)
                return;
            startReading();
        };

        startReading();
}
```

Figure 8: The actual reading from serial port <FloorControllerCSharp.cs>

The floor consists of eight flowers, as shown in the figure above. Each flower consists of four flower petals, and each petal has two parts (triangles). These triangles data is received in the <FloorControllerCSharp.cs> script. All the triangles of one flower are ordered clockwise.Each scene has a block of switch and if-cases to determine exactly which triangle is triggered.

User Interface

For the menu and highscore scene the button function is directly called, if the value of the touched triangle (on floor) is higher than the stepping threshold. To have an orientation a virtual floor pattern is created to see the location of the triangles in relationship to the Unity scene. Since there were problems with converting coordinates from different coordinate systems to world space, only functions were assigned to the specific triangles.

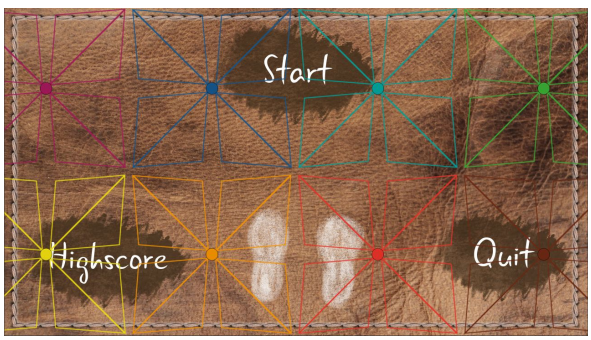


Figure 9: Virtual SensFloor flowers mapping to help design logic in game.

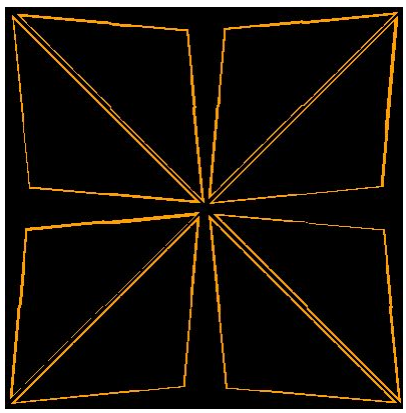
```
#region ChecksOnSpecificPartOfFlower
/*We are starting first row item from bottom left (1x1), Item labeling as "FloorFlower_Rows#_Cols#"*/
if (FloorCon == null)
    FloorCon = GameObject.Find ("_GM and Singleton Scripts").GetComponent<FloorControllerCSharp> ();

if (Floor.FloorControllerCSharp.GetStatusOfReadingData())
{
    if (FloorCon.FloorInput.x == 1)
    {
        switch ((int)FloorCon.FloorInput.y)
        {
            case 1:
                if (FloorCon.FloorInput.t1 >= FloorCon.StepValue || FloorCon.FloorInput.t2 >= FloorCon.StepValue
                    || FloorCon.FloorInput.t3 >= FloorCon.StepValue || FloorCon.FloorInput.t4 >= FloorCon.StepValue)
                {
                    highScoreBtn.Select();
                    PlayMenuSfx("Button");
                    Invoke("HighScore", delayTime);
                }
                break;
            case 2:
                if (FloorCon.FloorInput.t6 >= FloorCon.StepValue || FloorCon.FloorInput.t7 >= FloorCon.StepValue)
                {
                    highScoreBtn.Select();
                    Invoke("HighScore", delayTime);
                }
                else if (FloorCon.FloorInput.t1 >= FloorCon.StepValue || FloorCon.FloorInput.t2 >= FloorCon.StepValue
                        || FloorCon.FloorInput.t3 >= FloorCon.StepValue || FloorCon.FloorInput.t4 >= FloorCon.StepValue)
                {
                    leftFootBtn.Select(); //just highlighting foot
                    PlayMenuSfx("Left");
                }
                .
                .
                .
```

Figure 10: Reading floor data and check which triangle has StepValue greater than threshold

The data from the <FloorControllerCSharp.cs> contains the *FloorData* structures which can be easily accessed from a property called FloorInput. The structure *FloorData* contains variables containing position x,y and the pressure value of each triangle in a flower. Note that FloorData represents information for only one flower. In Figure 10, the sample code shows the use of FloorData in a <MenuController.cs> class, which selected the UI Button when the player foot touches the specific part of flower. Its bit of a manual work here to determine which triangle of flower is exactly under the UI button that you intend to do something with.

Sound (Feedback)

In order to understand better the data coming from floor, a sound feedback game is developed which uses the virtual floor mapping in Unity as play ground.



When player walks on the floor the triangles that are touched by player foot are highlighted visually and a random piano node is played. This feedback helped to evaluate the response time and this can also be used to check how sensitive the floor is in reading data.

Figure 11: Sound (feedback) Triangle no. 8 of Flower <2,3> is highlighted

3D Runner

The 3D in game scene behaves differently in comparison to the approach mentioned above. In the 3D Runner scene there are two buttons for sidestepping with a similar handling and also a walking gesture is implemented. In order to walk inside the game, the player steps on the feet shown on the floor and simply walks. Every time a foot is lifted in the air, the underlying flower does not send any data. This principle was used as a basis for implementing the walking gesture, using the amount of time that passed in between the receipt of data from the tactile floor. There is one timer for each foot which increases in every update loop. When a foot has contact with the floor the timer gets set to zero. If one foot is lifted in the air and one foot has contact with the floor, the algorithm recognizes this motion as walking and moves the character in game. The hover time for one foot is set to 0,5 seconds. The walking has a timeout of one second. Even when the player stops walking, the character moves until the timeout time is reached. This was added to prevent stuttering in-between walking steps.

There is an *issue* with this approach that when seesawing with your feet the floor still senses the movement and game recognize it as walking gesture.

2D Fish Pond

For the 2D pond scene there is a different, higher threshold for the receipt of usable data which needs to be reached. This should symbolize stomping to break the ice. Every time stomping happens, the underlying ice gets a bigger crack until it breaks. It is basically the same function as a button triggering.

Multi-display control

There are two projectors involved in the development of the game as the display hardware. One projector is dedicated to project on the wall, and the other is projecting on the floor. The details of the setup for the projectors are explained in the SRS document.

Figure 12: Projector setup in the laboratory

Both the projectors are controlled via Unity interface. During the development, the displays properties need to be set to show which display will project what particular part of the game. In order to achieve this, the features are divided according to the use of projector displays.

Display 1

This display is dedicated to projector features on the wall, as the content projected on the wall needs to have attention of player all the time. In Unity editor, the camera properties from the inspector need to be set; such as **Target Display** should be set to <Display1> for **Main Camera** of player and the **Canvas** of UI (see Figure below for details).

Figure 13: Main Camera settings for Target Display.

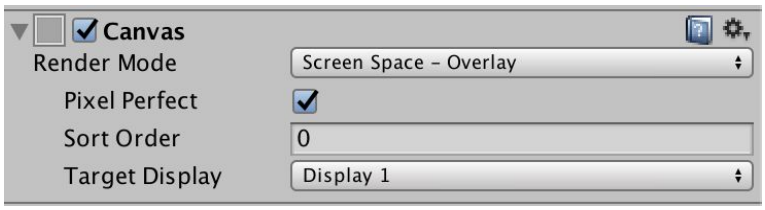


Figure 14: Canvas UI settings for Target Display.

Display 2

The major input source of the game is the SensFloor on which the game's main UI will be projected using Display2 as the Target Display from Unity editor settings. This type of UI is designed accordingly to fit to the size of the floor and is only projected with a resolution that matches the size of the floor e.g. 1280x720 pixels in our case as the projector has fixed zoom. So, the virtual floor is also a cutout as shown in Figure 9 and 11.

Figure 15: Snapshots from V1.0 of game prototype with SensFloor and projection of Main Menu

Player Controls

The player follows a dog/wolf which is used in the prototype instead of the dog model.
Player follows his companion wolf on his way back to the final destination which is a camp.
The path that player has to follow is generated randomly using <SplineController.cs> script.

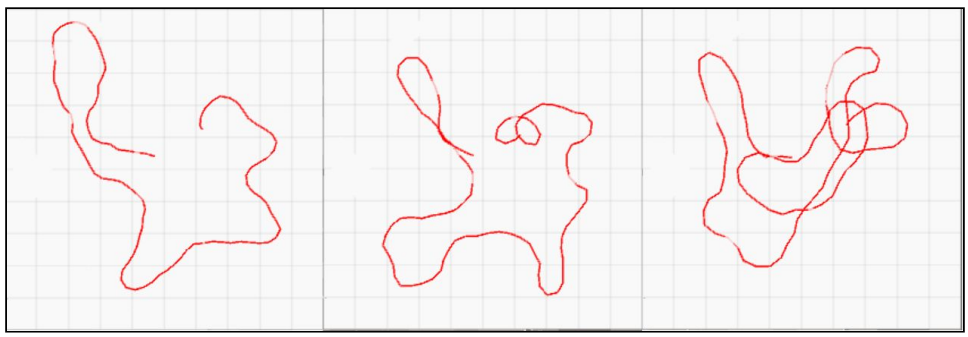


Figure 16: Random path generation for player to follow

The starting and ending of path are not same and the end of path node is fixed to the position of camp. The script is attached with both player and wolf, as the both should follow the same path to reach end point. The parameters of the path can be tweaked and its length can be changed as well, which will result in changing the overall time of level.

The wolf runs in front of the player with a fixed distance and has two animations. One is when player runs, the wolf starts *Running* as well and when the player stops the wolf goes into *Idle* animation.

The player movements are controlled by <PlayerRailController.cs> script, which deals with the floor data input and sets the game states (None, Idle, Runner, LeftStrafe, RightStrafe, FishPond2D, QuizScene, ...etc) as well according to actions and trigger during gameplay.

```
public void GetPlayerInputFromFloor ()
{
    if (FloorCon == null)
        FloorCon = GameObject.Find ("_GM and Singleton Scripts").GetComponent<Floor.FloorControllerCSharp> ();

    _leftFootDecay += Time.deltaTime;
    _rightFootDecay += Time.deltaTime;
    _walkingTimeOut -= Time.deltaTime;

    if (FloorCon != null) {
        /*We are starting first row item from bottom left (1x1), Item labeling as "FloorFlower_Rows#_Cols#"*/
        if (FloorCon.FloorInput.x == 1) {
            switch (FloorCon.FloorInput.y) {
            case 1:
                if (FloorCon.FloorInput.t2 >= FloorCon.StepValue || FloorCon.FloorInput.t3 >= FloorCon.StepValue) {
                    this.PlayerStrafeLeft ();
                }
                break;
            case 2:
                if (FloorCon.FloorInput.t1 >= FloorCon.StepValue || FloorCon.FloorInput.t2 >= FloorCon.StepValue ||
                    FloorCon.FloorInput.t3 >= FloorCon.StepValue || FloorCon.FloorInput.t4 >= FloorCon.StepValue) {
                    this.LeftFootActive ();
                }
                 if (FloorCon.FloorInput.t6 >= FloorCon.StepValue || FloorCon.FloorInput.t7 >= FloorCon.StepValue) {
                    this.PlayerStrafeLeft ();
                }
                break;
            case 3:
                if (FloorCon.FloorInput.t2 >= FloorCon.StepValue || FloorCon.FloorInput.t3 >= FloorCon.StepValue) {
                    this.PlayerStrafeRight ();
                }
                if (FloorCon.FloorInput.t5 >= FloorCon.StepValue || FloorCon.FloorInput.t6 >= FloorCon.StepValue ||
                    FloorCon.FloorInput.t7 >= FloorCon.StepValue || FloorCon.FloorInput.t8 >= FloorCon.StepValue) {
                    this.RightFootActive ();
                }
                break;
            case 4:
                if (FloorCon.FloorInput.t6 >= FloorCon.StepValue || FloorCon.FloorInput.t7 >= FloorCon.StepValue) {
                    this.PlayerStrafeRight ();
                }
                break;
            }
        }

    }
    //left feed on ground -> triggers data, right feet in air not -> walking. || other side
    if ((_leftFootDecay == 0 && _rightFootDecay >= _timeWithoutContact) || (_rightFootDecay == 0 && _leftFootDecay >= _timeWithoutContact)) {
        _isWalking = true;
        _walkingTimeOut = _walkingTimeOutMax;
    }
    //if player stands still (and no data triggers) or is triggering data too fast, he is not walking and the timeout happens
    if (_walkingTimeOut <= 0f)
        _isWalking = false;

    //TODO: is walking feet highlight in UI
    if (_isWalking) {
        this.PlayerMoveForward ();
    }
    else if (!_isWalking)
    {
        this.PlayerStopMoving();
    }
}
```

Figure 17: Movement of player with floor data input <PlayerRailController.cs>

CONCLUSION

The game prototype has completed all features as discussed and designed in the feature list. It gets the data from the SensFloor and updates the player position accordingly in the gameplay. The interaction with other UI elements also works and are implemented with visual and sound feedback.

The game works well with the SensFloor but has some lag between the response time and data stream from floor. Even if the player leaves the floor, still there is some data coming as noise. The floor is really sensitive to movements around it.

There are some overall issues which can be improved in game prototype and SensFloor implementation.

a. Serial communication is a bottleneck for big floor. If the size of the floor increases there will be more lag in reading data.

b. The floor data is read as a continuous stream and there is no reset of data when the scenes are switching. Hence, it leads to old data being used in the new scene.

c. The data from floor is highly sensitive even if player steps only once, the floor sends a stream of data for that single event and that in case of playing sounds causes issue.

d. A part of path is generated outside terrain bounds. Since, its random generation the path if is generated close to terrain bounds will let player see the beyond terrain.

e. There is lag in walking gesture, since it’s not the best solution to control walk movement with time.

f. The project on the top of floor causes strong shadows on floor, which in result hides the actual projection from game.

Project Management

The project management was all carried by using Basecamp 3 <https://3.basecamp.com/3360695/projects> all the team members and supervisor have access to the project related tasks and document. This tool helped the team to communicate about the meetings, task management and document management. For the code, git repository was used which also shared with Prof. Hansen.

The timeline for the DE project was from Apr'16 to Feb'17. In this duration, two different teams were assigned with me to finish the project tasks. Here are the details of the project team members and duration of their availability.

Team 1 (Apr'16 - Oct'16)	**Team 2 (Nov'16 - Jan'17)**
Developers: *Asema Hassan (Project Leader) *Marleen Rohde *Michael Kropp	Developers: *Asema Hassan (Project Leader) *Stefan Schwarz *Kay Illner
Designers: *Nicolas Pepping *Mareike Gabele	Designers: *Robert Wlcek *Stefanie Vogel

The division of major tasks over the duration of project and improvements done before the final submission of DE project.

Team 1 (Apr'16 - Oct'16)	**Team 2 (Nov'16 - Feb'17)**
Developers: *C++ Plugin *3D Runner gameplay *FishPond 2D *Menu *Highscore	Developers: *CSharp Plugin *Scalable to any floor size *Improved 3D Runner *Path Following *2D - Sound feedback *Player Gesture Controls *Quiz Learning *3D Models for items (quiz,fish, berries) *UI feedback elements *Sounds integration
Designers: *Story *Basic HUD *Sounds *Menu *Highscore	Designers: *New HUD *Menu *3D Terrain *3D models *UI elements *Sounds SFx/VO

Thanks to all team members who were involved in the process to finish game prototype.