Creating a Multi-Touch Game

Pontus Werme
2015

Bachelor of Science in Engineering Technology
Computer Game Programming

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Creating a multi-touch game

Author: Pontus Werme
pwower-2@student.ltu.se

Supervisor: Patrik Holmlund
patrik.holmlund@ltu.se

Spring 2015
Luleå University of Technology, Skellefteå
Department of Computer Science, Electrical and Space Engineering
Abstract

This project develops a game that can be run using Nuiteqs engine called Snowflake. Snowflake is a touch interface engine, used for screens with touch sensors. The game which I choose to develop for this project is called "Sheep Game". "Sheep Game" is developed using C++, and the level design with XML. The game has multiple levels and has a puzzle aspect. When playing the game you control dogs that are trying to get the sheep's into their pasture. First the different touch technologies will be presented, then how my game is designed and how it works.

Sammanfattning

Acknowledgement

I take this opportunity to express gratitude to all who helped me. Major thanks goes to my supervisor Patrik Holmlund for helping me whenever i had questions, aswell for his comments on my thesis drafts; big thanks also goes to the employees at Nuiteq whose valuable feedback helped a lot during the development process and for letting me do my work there.
Thanks as well, to the people who helped me test the game, which helped rebalancing the game and of course, another thanks goes to the employees of Luleå University of Technology for all of their support throughout my education.
Abbreviations and Terms

XML - eXtensible Markup Language - a file format used for creating documents in a readable format, both for the machine and for users.

AI - Artificial intelligence - is the intelligence exhibited by machines or software.

2D - Two dimensional environment
1 Introduction

Touchscreen technologies

A touchscreen is where the display also is an input device, meaning that pressing on the screen will interact with the computer just like a mouse interacts. Touch based interaction have become ever more popular over the last ten years, mostly due to the increase in smartphones and tablets[1], but also in the medical field[2] simply because it’s easy to use and keep clean(it’s safe to clean).

Touch interaction is where the inputs registered with touches, which is received when the user places a finger(or object) on the touchscreen[3]. This interaction method with the application could be more complex to use (doing something where it might require two or more touches), but not necessarily harder to comprehend, e.g most people know that placing two touches and pulling them away from each other zooms in on the application. That gives the developer the option of using many different types of input(amount of touches, their direction and/or touch size).

There are four main groups of touchscreens:

Capacitive

The first touch screen was a capacitive and was developed by E.A Johnson at the Royal Radar Establishment, Malvern, UK in 1965-1967[4].

![Image of capacitive touchscreen](image)

Figure 1: A finger disrupts the current and is registered as a touch.

The capacitive touch screen is a single glass layer coated with a conductive material applied on it. This single layer measures conductive materials touching the screen, meaning that non-conductive material won't register a touch, see Figure 1. This is a durable and energy efficient screen since it only uses one layer and therefore is the most popular choice among manufactures[5]. Can be found in most smartphones built after 2006 (e.g. iPhone[6] ).
Optical(infrared)

Infrared touch screens are based on light-beam interruption technology. It's built by a grid of infrared LED and photodetectors, where beams go horizontal and vertical. Pressing on the screen disrupts the beams which is registered as a touch. It registers any kind of touch, be it conductive or not; see Figure 2. The advantage is that this type gives the best image quality of the available options[7]; the screen have a much higher life span, since the light only needs to be disrupted: a strong outer layer can be used. Disadvantages is that it's sensitive to ambient light. Accidental activation from dirt and similar substance. The screen have a higher manufacturing cost than the other technologies.

Resistive

The resistive screen is built by two layers; uses a flexible top layer and a harder bottom layer, both are coated with conductive materials, between these layers is a gap. Would the screen be pressed the gap would close and a current would flow between the two layers. The position of the touch is calculated by measuring the voltage[8], see Figure 3. The advantages of this is that is low power consumption, can be activated by any kind of touch (e.g. pen, finger) Disadvantages is a lower image quality since the image is projected through the two layers, the top layer is vulnerable to scratches och other piercing damage.
Surface Acoustic

The screen is built with a single glass sheet with a grid (just like the optical) of ultrasonic waves on the surface.

![Figure 4: A finger disrupts the wave grid and is registered as a touch.](image)

Touching the screen absorbs the sound waves resulting in a drop of amplitude, see Figure 4. The advantage with this technology is excellent image quality[9], since it only needs the glass sheet to project sound on. Disadvantage with the surface acoustic is that it don't registers items that don't absorb sound well like hard items e.g. pen, fingernail, it need recalibration periodically. Any type of dirt or liquid will affect the result, and might even register as a touch.

Nuiteq

Nuiteq is a software company that develops the Snowflake engine; a touch interaction based engine for PC, Mac and Android. Nuiteq is based in Skellefteå, Sweden and Bangalore, India. The Snowflake engine is developed for education, business, and/or games in mind. It's designed to help the business with a multitude of businesses applications, teach others with interesting and helpful learning tools and play games that's fun and exciting. Snowflake could be used by teachers in schools to make teaching easier. This report is based on the goal of developing a game for Nuiteq's Snowflake engine.[10]
1.1 Background

Nuiteq wanted more applications for their Snowflake engine. They are in the process of developing their own tools and games that will be included in the next engine release, but however they wanted more applications to be developed, so they looked for outside help.

The aim of the project was to develop a game that Nuiteq could use in their engine, a game that is designed for multi touch in mind; to demonstrate the Snowflake engine. The application was to be developed in Nuiteq's headquarters close to LTU Skellefteå, Sweden.

1.2 Third party software

During development, third party software was used to make the development easier/possible.

1.2.1 Snowflake

Nuiteq's Snowflake engine is a cross platform engine (PC, Mac and Android). Which is designed for touch screen of various sizes. Using Snowflake saved time during development, it already contained elements such as rendering of images (and other UI elements) and input handling.

1.2.2 Qt

The Qt ("cute") library is a cross platform library written in C++, originally designed for UI, now used for e.g. XML, Databases, multimedia, OpenGL, scripting and more.

"Qt is created by developers for developers where making developers lives easier is top priority." - Qt development team.[11]

1.2.3 Box2D

Box2D physics engine was a easy choice, since having previous work experience with it. Box2D is an open source C++ physics engine for simulating rigid bodies in 2D. Box2D is developed by Erin Catto and has the zlib license[12]. Box2D worked well for creating boundaries (for e.g. pastures and borders for the level) and sheep physics interaction with the world.[13]
1.3 Goal

The goal of this project is to develop a game that use the Snowflake and is fully integrated with touch interface in mind.

The following bullets was needed to create the game:

- Learn the snowflake api (application programming interface), How the engine setup and how a player interacts with a game.
- Implement AI for sheep and dogs
- Intresting puzzle aspect that the player must solve
- High Score and a time limit to motivate the player to work fast and efficient.

1.4 Limitations

The time of development is a mere eight weeks, resulting in some aspects of the game could not be implemented. Such as a versus mode and a level editor.

The sheep AI was written a bit simpler that it was thought out to be, since it did not function in a way that was fun (Sheep moves as a group in real life, almost glued to each other)\[14], which removed the point of having multitouch when one dog could move a whole group(The design was that they would move away from dog regardless of other sheep position).
1.5 Method

Developing the game in Visual Studio 2013, using the Snowflake engine. Testing is being done on a local computer with touch interface. The development was split into multiple development phases:

1.5.1 Research and development phase

This phase consisted of planning out the game design and researching how the engine's API worked. To start of installing Snowflake development kit and reading the documentation was essential. Along with the Snowflake development kit came a few test application that gave a helpful nudge in to how to develop applications for the engine. Nuiteqs code standard of Snowflake was essential to learn since Nuiteq regularly looked through the code to find flaws and/or places that could be improved. The game design can be found in the appendix.

1.5.2 Developing phase

When the design was planned out and the test applications have been fully explored; It was time to develop the game. A day to day production blog can be found in the appendix.

1.5.3 Testing phase

Testing the application to rebalance a few things was key to get the feel of the game right. To my help doing this I had some fellow students and, employees of Nuiteq.
2 Design and Implementation

2.1 Game design

Game name: Sheep Game
The game world is an open field; with a few empty pastures. In the open field sheep are roaming around and it is the player(s) goal to spawn dogs and use them to heard the sheep back into their pastures. However this is not so simple when the sheep have a tendency not to stay in one place. The game would feature a puzzle aspect, and have multiple levels with increasing difficulty. It’s aimed at an international user base, men and women in all ages. The game contains a competitive aspect; when the player finishes a level they have the option to save the score if their score was good enough to be saved. Score is stored locally and for each level.

When a touch begins a dog will be spawned, when the same touch ends the dog leaves. Moving the touch on the screen will move the dog the same path. When spawned the dog runs in from the side and starts to bark when it’s getting close to the touch, the opposite thing happens when the touch ends; the dog runs of the screen. The dog can be picked up once it’s leaving by another touch.

Dog barks to make the sheep move away from the it. The goal of the game is to herd the sheep back into the correct pasture. The pastures have limited slots for the sheep to enter and can only care for sheep of a certain fur color. The limit of dogs on the screen is only limited on amount of touched that can be placed.

Obstacles will be in the levels, trees and rocks; Anything that will make the map less dull och more fun. The difficulty increases as the player progresses throughout the game: with more sheep, smaller pastures, different sheep colors and more obstacles(trees, rivers etc). There are five types of sheep and pastures: black, blue, red, white and yellow.

If a sheep enters the wrong pasture the time adds two seconds to the current level duration. Giving the player a reason to choose to do the herding carefully receiving a larger time with few penalties or do it fast herding them from pasture to pasture getting a lower time but more penalties.
2.2 Application

![Game States Diagram]

Figure 5: Game States: Arrows represent state switches.

**SheepGameApp**

Core application that contains crucial information to make the game able to run in the snowflake engine. The application uses states to easily separate assets where it’s needed, to have an efficient runtime. Contains a state machine to easily go between the states the game can be in. For state changes see Figure 5.

**TitleScreenState**

This state shows the title with a scrolling background. The player will have the option to launch the first level (and go to **gamestate**) or choose level select to switch to the **LevelSelectState**.

**LevelSelect State**

This state shows all the available levels, pressing any of them will switch to **GameState**.

**GameState**

Entering this state will load the level what the user choose to play. This state contains the managers and is where the game is played, the user can restart or go back to **TitleScreenState** by using the menu buttons. Finishing the levels will change state to **LevelCompleteState**.

**LevelCompleteState**

This state will give the player the option to enter a new highscore if the player score enough to be added on the list. Otherwise the player will see their score and a few statistics that was tracked during game play (e.g. Amount of dogs spawned, Dog distance moved).
2.3 Managers

2.3.1 Entity Manager

Entity manager is where the entities are stored; When the application registers a touch its forwarded here to determine if a new dog is to be spawned or an old one picked up. Moving the touch requires checking the touch unique id to find the correct dog associated with that touch. Once the level is loaded using the Level Loader class the level information is stored in this struct.

2.3.2 Physics Manager

Wrapper for the Box2D physics Engine. The sheep, obstacle and boundaries all use Box2D physics objects which is stored and used in the Box2D world during gameplay. The Physics Manager updates the position and velocity of all entities each frame.

2.4 Tools

2.4.1 Level Loader

The level Loader is a class with only static functions. The class uses QT's QDomDocument[15] as the XML reader. Its saves the sheep's, pastures, and obstacles in individuals data storage blocks while the remaining information is saved in a struct called LevelInfo, which stores information that is only needed to be specified once(e.g. sprite name, sprite size)
2.4.2 XML layout (Level Layout)

```xml
<Sheep_Game>
  <Level_Setup LevelBackground="background1.png">
    <Level_Setup Scale="0.12" MaxSpeed="0.3" Sprite="SheepSprite"/>
    <Dog_Setup Scale="0.06" MaxSpeed="0.003" BarkDistance="0.2" Sprite="DogSprite.png"/>
    <Frasure_Setup PastureSprite="Frasure" GateSprite="Gate"/>
  </Level_Setup>
  <Level_Pastures>
    <Pasture Scale="0.4" MaxSheep="2" Gate="Right" PositionY="0.5" Color="Red" PositionX="0.148"/>
  </Level_Pastures>
  <Level_Sheep>
    <Sheep Rotation="-110" PositionY="0.219009" Color="Red" PositionX="0.743598"/>
  </Level_Sheep>
  <Level_Obstacles>
    <Obstacle Texture="tree1.png" PositionY="0.8" ImageScale="0.2" PositionX="0.3">
      <SphereCollider Radius="0.3"/>
      <RectangleCollider SizeX="0.1" SizeY="0.1"/>
    </Obstacle>
  </Level_Obstacles>
</Sheep_Game>
```

Figure 6: Layout from the level XML file.

**Level_Setup**

Prevents a lot of repeats (e.g. since all sheep will share scale, maxspeed and base texture they only needed to be specified one, see figure 6).

**Level_Pastures**

 `<Level_Pasture>` defines one pasture in the level. Here the pasture have it’s location, scale, color and gate location specified with an enum(Top, Left, Right, Bottom or Random).

**Level_Sheep**

 `<Level_Sheep>` defines one sheep in the level, the sheep are predefined in the level and not random to give all users the same chance on each level.

**Level_Obstacles**

 `<Level_Obstacles>` defines one obstacle in the level, the obstacle can have a collider tag with have PositionX and PositionY attributes to specify its position, otherwise it will use the obstacle position.
2.4.3 Query Callback

This class inherits the Box2D callback class so it can be used by the Box2D world for the sheep to detect boundaries or obstacles around them. They use this to raycast in front of them.

2.4.4 Contact Listener

This class is called when the Box2D world finds a collision. Whenever a sheep enters a pasture the Contact Listener tells the pasture if it receives a correct sheep or a wrong one. It also handles sheep collisions (with other sheep and boundaries) so they stop moving when collided with something.
2.5 Entities

2.5.1 BaseEntity

A base class for all entities to make storage and managing the entities easier and more efficient.

2.5.2 Dog

A dog is created when the user touches the screen, however the dog is spawned on the closest edge of the screen. This is to motivate the user to switch dog less often (keeping the same touch). Moving the touch will move the dog, the dog have a maximum speed however and will try to catch up to the touch if it moves too fast. The dog barks once it's close enough to the finger which will make the sheep move away from it in the direction of the bark.

![Image of a dog surrounded by sheep]

Figure 7: When dogs is barking, sheep will try to keep a distance from it.

When the player eventually release the touch the dog runs in the direction it's facing, however the dog can be picked up again if pressed before being outside the screen where it will be deleted.
Dog States

The dog have two states *DogControlled* and *DogLeaving* (not controlled), see fig 8.

*DogControlled*
Dog is moving towards the current touch that matches its ID, in this state the dog is barking (however not before it have caught up to the touch for the first time).

- Will not however start bark if the touch is too far away.
- Letting go of the touch will make the dog enter *DogLeaving* state.

*DogLeaving*
The dog is no longer barking.

- Dog moves in the direction it last faced and will exit the screen.
- Making a new touch close to the dog will return it to *DogControlled* state.

Figure 8: Dog states.
2.5.3 Sheep

Sheeps are created when the level starts and will remain in game until the user exits the level. They mostly stand still when left alone, but occasionally they will randomly wander around to find a new spot to eat(idle).

Sheep States

Sheep have three states: *Idle*, *SeekNewTarget* and *RunningFromDog*, as shown in fig 9.

- **Idle State**
  - This state is where the sheep stands still for a random amount of time depending on where the sheep are:
    * Out on the field: 3-10 seconds.
    * Inside correct pasture: 4-15 seconds.
    * Inside wrong pasture: 2-6 seconds.
  - After the idle time is over the sheep will enter the *SeekNewTarget* State

- **SeekNewTarget State**
  - This state is where the sheep moves to a random location:
    * Out on the field: A random position that is in its field of view(90 degrees to its left to 90 degree to the right). It will raycast to see if any boundary is in the way, if one is found it will choose the location that is on the opposite side from the raycast(e.g 72 degrees to the left to 72 degrees to the right) and use that(only raycast once to save performance).
    * Inside correct pasture: If the sheep would find itself inside a pasture that is the correct color and have a slot for it, the random location is inside the pasture regardless of the sheep's view. When a sheep is inside a correct pasture, they can't leave(there colliding physics is turned of so they don't collide and accidentally push each other outside the pasture and they dont react to dog barks).
    * Inside wrong pasture: The sheep can also happen to be inside the wrong pasture; if the sheep have the wrong color or the pasture have no more slots open, it will find a location just outside the gate.

Once it have found a location, it will move to it. If a sheep in this state collides with anything it will go to the *idle* state.

- **RunningFromDog State**
  - This state is automatically entered if a dog is barking on the sheep for any state, not however is the sheep is in the correct pasture
Sheep State Transitions

Figure 9: Sheep state changes

1. When the idle timer is zero it's time for the sheep to find a new spot to move to.

2. The sheep reached the location, hit something or the move timer ran out(move timer is to prevent the sheep of getting stuck)

3. A dog is barking on the sheep, it will immediately try to run away, unless inside correct pasture.

4. Return to idle state when sheep is outside of the barking range.
2.5.4 Pasture

The pasture consist of two sprites, the pasture and gate sprite. The pasture contains five bodies: three fences, one gate, and one sensor that is contained inside the pasture.

![Figure 10: A pasture from the game.](image)

The blue lines with black border are static boundaries, the striped box is the gate which will spawn once the pasture is full. The green area is a sensor, which will detect any incoming or outgoing sheep. The number in the center represent current sheep in pasture vs max amount of sheep which is allowed in the pasture. The size of the pasture and gate position is defined in the level file. When a sheep enters the pasture the sensor will be alerted that something has entered it. Its going to check if it was a sheep, then it will check if the sheep color matches with the pasture's color, which in this case is blue. Same color sheep will want to stay in the pasture while wrong colored will want to leave.
2.5.5 Obstacle

A simple level object with a sprite and physics. The obstacle physics can be specified as either a sphere or a rectangle. The obstacle physic will count as an obstacle so all sheep can collide with it.

2.6 Testing

The game has been tested continuously throughout the entire course of the development. Dog and sheep movement was thought to pinpoint, since it must give a good feedback to the user. During testing minor details was discovered that helped the gameplay (e.g include a timer and add penalties when a sheep entered the wrong pasture)
3 Results

The game came very close to the planned result. The game will be presented with six levels with increasing difficulty. A level selection screen and highscore list which saves scores locally. The final product have a fun gameplay, suited for all ages.

The application saves highscore locally in temporary storage. A highscore of up to three is saved where the user with the least time to complete the level gets a chance to save their score along with a seven character long name.

The application have a lot of replayability since one can always improve their score. Multiple people can play at the same time cooperating to get a good time as possible.

The average play time from first level to last is close to 20 min, that being if they play each level once.

Since the development team only consisted of a programmer, the look of the game is somewhat lacking. The title screen is lacking background sprites fitting the applications art design, as well as the buttons.

Test users have rarely given up during gameplay, they always wanted to finish the level. The hard levels have just enough frustrating tendencies as the rewarding ones.

3.1 Future work

Due to my lack of graphical designing skills most graphical assets of the game looks plain. The entities was planned to have animated sprites but not being able to find any I liked the static image was chosen.

More levels and have changing pastures (switch color during gameplay) so you have to time it would give a fun experience.

A wolf that occasionally enters the game to steal a sheep was planned but scrapped due to lack of time. I wanted pastures to have dual sprites, one below sheep and one above to give the feeling of 3D in the game(e.g. a roof somewhere in the pasture).

3.2 Social, Ethical and Environmental Considerations

The gameplay was designed to be very easy to understand and play. No help is needed when playing for the first time. Once the player places the first touch and sees the dog moving in and move the sheep, they will understand how the game is played. The game motivates teamwork making it a social game, suited for all ages. The game reward sorting sheep on the basis of their fur color which led some people to think of racial segregation, this was not the intent, so a few changes was made to the game. User information distributed from the application.
4 Discussion

Developing an app with touch interaction is sizable different compared to developing to a mouse and keyboard interface, or gamepad interface: Having an arbitrary amount of touches active on the touch screen, the application will have to handle them all individually where each could affect the application differently. The game state allows any number of touches, spawning a new dog for each one (to the limit of the hardware).

The other states only handles those who begin and end their touch on a button.

The user will having trouble seeing what they are interacting with since the touch (eg. finger or pen) is directly over the touch or the arm is covering a big part of the screen. Having bigger sprites solves some parts of this but since any number of players can be playing the sheep game difficulty seeing the screen is a real problem.

Only one type of interaction, a touch; whereas on a computer the user have a mouse with right and left click, and a keyboard, the game was designed with this in mind. Having buttons that on a keyboard and mouse game would not have been necessary but on a touch interaction game was added (e.g. Back-to-menu button where on a keyboard the ESC key would have sufficed).

A few things changed during development due to the technical limitations of Snowflake, e.g. the highscore had to be saved in another place since snowflake by default do not run with administrator rights.

5 Conclusion

The project was a success, the app run smoothly (even on slow computers). Throughout the project there was very few things that halted the development, otherwise there was pre planned solutions to most issues that could arise. If Nuiteq would choose to use this app in a future release of Snowflake, they would hopefully find it a good contribution to their product.
References

[1] Touch-screen phones poised for growth
   http://tinyurl.com/oypjqlq
   Published by May Wong, AP Technology Writer.

[2] RELIABLE TOUCH SCREENS FOR MEDICAL APPLICATIONS
   http://tinyurl.com/ngh85sd
   Published by Interelectronix

   http://en.wikipedia.org/wiki/Touchscreen

   http://tinyurl.com/3u97zyj
   Published by Mary Bellis, Inventors Expert

[5] Touchscreens or Human Machine Interface (HMI)
   http://tiny.cc/o7myx
   Published by EngineersGarage

   http://en.wikipedia.org/wiki/IPhone#Screen_and_input

   http://tiny.cc/tcmyx
   TRU-Vu Customizable Industrial LCD Monitors

[8] Resistive Touchscreen: Is Resistive The Technology For You?
   http://resistivetouchscreen.org/

[9] Surface Acoustic Wave Technology (SAW)
   https://www.touchsystems.com/surfaceacousticwave
   Published by TouchSystems Corporation.

[10] Snowflake homepage
    http://nuiteq.com/

    https://www.qt.io/about-us/

[12] Zlib licence
    http://en.wikipedia.org/wiki/Zlib_License

[13] Box2D homepage
    http://Box2D.org/

[14] Sheep Movement example
    https://www.youtube.com/watch?v=bpjP3mxv2Is

[15] QT XML reader
Appendix

Game document and production blog

Time Plan, game design document and can be found here:
www.tinyurl.com/sheepgame2015
Day to day production blog:
www.tinyurl.com/sheepgame2015blog