1. Introduction

Air hockey requires an air-hockey table, two paddles (Mallets), and a puck. The table consists of a large smooth playing surface, a surrounding rail to prevent the puck and mallets from leaving the table, and slots in the rail at either end of the table that serve as goals. Additionally, tables will typically have some sort of machinery that produces a cushion of air on the play surface through tiny holes, with the purpose of reducing friction and increasing play speed.

QAirHockey is an idea drawn from the air hockey concept but extended to the mobile devices. However to encourage more people to play at the same time, QAirhockey combines numerous playfieds to create a larger hockey field than standard. The puck and mallet in QAirhockey may move between different players mobile device screens. Figure 1. Depicts the playfield scenarios.

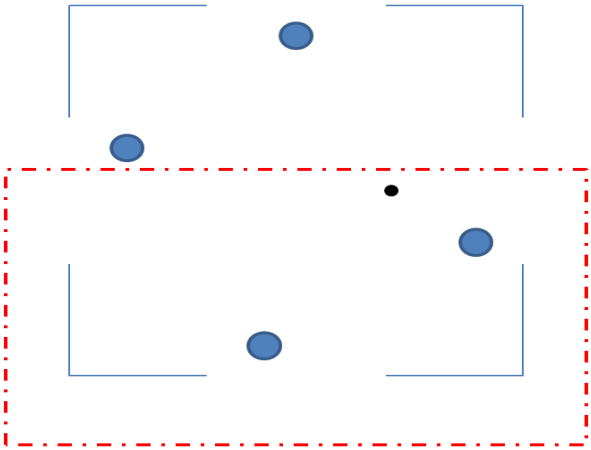

Figure 1. A playfield with four players and one puck. View of player 1 in red square.

2. Basic requirements

2.1 Initialization

Initializing the QAirHockey game will involve players joining the Local Area Network (LAN) and consequently joining a network broadcasted game. The typical scenario architecture is depicted in Figure 2.

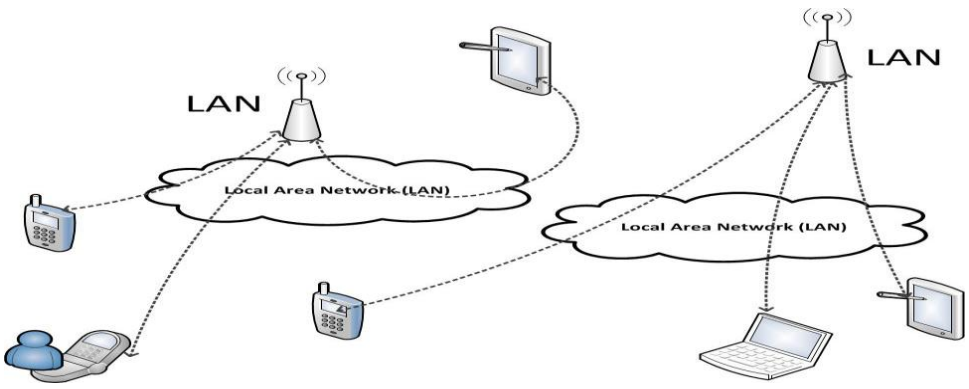

Figure 2. Typical game scenario with two LANs and six players.

Bishal Karki, Tommi Kallonen, Were Oyomno {firstname.lastname}@lut.fi

2.2 Game flow

QAirHockey aspires for a basic and simply enough flow in game playing. This flow consists of the following requirements.

1. Player joins the Local Area Network (LAN).
2. Player starts the game and receives a life value of 5.
3. Other players join the game receiving similar life values of 5.
4. The game starts and players move their mallets by tilting the mobile device
5. Players shield their goal using the mallet and try to direct to opponents goal.
6. Conceding a goal results in one's life deducted by 1.
7. When a player's life value reaches 0 the player is ejected from the game.
8. The last remaining player wins the game.

2.3 Game state machine

Based on presented game flow a finite state machine (FSM) of the QAirHockey game events are depicted in Figure 3

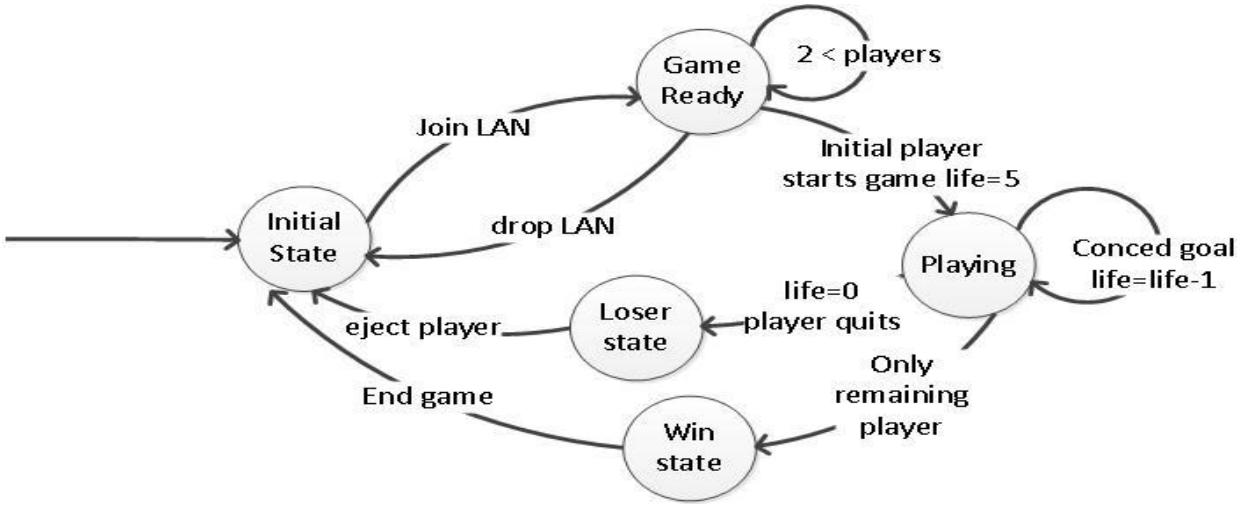

Figure 3. Finite State Machine of QAirHockey stages.

3. Designing QAirHockey

The design process of QAirHockey is documented in the stages undertaken during the Qt codecamp. These stages detail functionality realizations regarding the classes, start screen, game playing, game ware (mallet, puck), the playfield, goals and winning decisions.

3.1 Classes

The QAirHockey is split into five different classes implemented on this codecamp. It also uses four classes from network chat example from QT toolkit examples with minor changes.

Table 1. QAirHockey classes

Class	Purpose
Startscreen	The start screen with logo. The game gets started here
Game	Handles the actual game. Reads user input and calculates puck and mallet movement
Mallet	Handles the position and drawing of a mallet
Puck	Handles the position and drawing of a puck
Playfield	Handles the drawing of playfield

3.2 The Startscreen class

This creates a visible starting point "start screen". In this stage the new game instance is created but the actual game playing does not begin until a number of persons (2 - 4) have joined the game and the initial player wishes to start the game/tournament.

3.3 The Game class

This the main class of the game. The game class takes input from user and network and handles the physics involved in the game playing session. As the game is created, it also creates:

1. A client for networking
2. Playfield
3. The relevant number of pucks used in the game
4. The correct number of mallets to be used in the game depending on the number of players.

As the game is started the pucks and mallets are placed in their start positions and pucks get random speed and direction on movement to start with. The game also sets the orientation and placement of the view, so each player sees the correct portion of the playfield in correct orientation (all calculations of puck and mallet locations are calculated using same coordinates on all devices so the view needs to change place and rotate).

During a game a function called updateGame is called periodically. This function reads the acceleration sensor values to calculate new speed of user's mallet and calculates new position of mallet also checking that the mallet remains inside the playfield. The puck locations are also calculated using puck's speeds in x and y directions. For the pucks collision detection is also done. First the game checks if puck hits a wall and if so, changes its speed according to the hit. Then it checks if pucks hit each other and again corrects the speeds if needed. Thirdly the game checks if a puck hits player's mallet. Again if this happens new speeds are calculated. If a collision happens between player's mallet and a puck, the game sends a message to all other players with the information about puck's new location and speed in x and y directions. In this manner, other players get correct information about this puck's location and calculate its location correctly from now on. The game also checks if a puck has gone to this player's goal. If this has happened game sends information about this to all other players and starts the puck again in the middle of the playfield with random x and y speeds.

As mentioned, the game sends information about puck after the player has hit it and about a goal. The game also sends a message periodically about the mallet location of the player so other players can also see the mallet if it happens to be in their view.

3.4 The Mallet class

This is a simple class charged with the responsibility of tracking the mallet's location. Additionally this class may issue a draw command when needed to update the mallet's current location.

3.5 The Puck class

To keep track of the puck(s) during game playing session, a puck class is defined. The puck is simple enough to keep track of a puck's location and draws it when needed. In many respects this this class is similar to the mallet class.

3.6 The Playfield class

This is slightly more complex class compared to the puck and mallet class. The reason behind its complexity is the reversing of position to appear relevant to different players' orientations. The playfield class is mainly used to drawing the playing surface for desired number of players.

Table 2. Classes from network chat example used in QAirHockey

Class	**Purpose**
Server	Listens for invoming connections
Client	Handles messages that are sent between devices
Peermanager	Creates connections with other devices
Connection	Handles individual connections

The classes used with networking in QAirHockey are mostly copied directly from network chat example. These classes can automatically find other devices with same program running by starting a listening server and broadcasting messages to see who responds.

The changes done to the example codes are mostly in Client class where a number of new messages have been added to send and receive information about puck locations, mallet locations and goals scored.

4. Implementation

To implement the QAirHockey, the Qt creator 2.0.1 and Nokia Qt SDK 1.0 and 1.0.1 were adopted by team members. Two notebooks were used to test the game deployment in simulated environment and finally two Nokia N900i devices were also used to test the validity of the implementation in real life deployment. Tests with a notebook and Nokia N900i were inconclusive.

During the implementation phase of QAirHockey a number of Qt platform widget features were utilized. The features provided functionalities for varying gaming aspects. This section will details some of the utilized features, screen shots from the QAirHockey realized and detail some challenges discovered in the process.

4.1 Qt platform APIs

In the implementation 5 Qt platform APIs were utilized. These included: (1) **Qt Networking** – Enabled devices to connect and exchange information over LAN, (2) **Qt Sound** – Sound effect during starting game, collision and end, (3) **Qt Sensors** – Tilt/Accelerometer sensor to move

players' mallet and puck, (4) **Qt Mobility** – Enable other sensor like haptic feedback / vibration and (5) **Qt Graphics** – To draw puck and other needed features.

4.2 Screenshots

Some visualization to the realized implementations of QAirHockey, are depicted and shortly explained in caption in Figures 4 (a), (b).

Figure 4 (a) Startscreen.

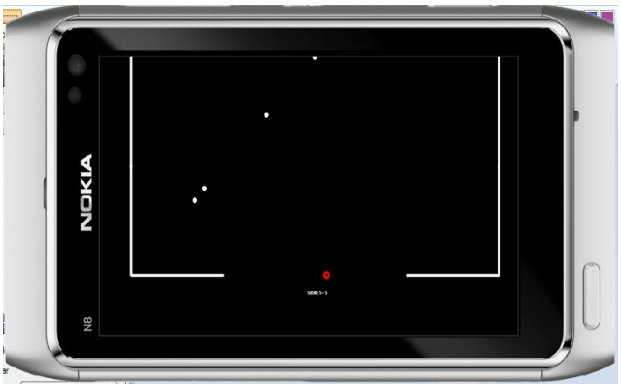

Figure 4(b) Game playing session.

4.3 Challenges

A number of challenges materialized during the implementation of QAirHockey. Whilst some of these challenges were solved to a certain degree others remained to the end however would be solved in the next iteration of the QAirHockey implementation. The main challenges in include:

1. Difficulties in installing **Qt Mobility 1.1 Beta** – With this framework more advanced platform features of Qt would have been realized. Such features are the haptic feedbacks, vibration, camera, contacts APIs, maps navigation and so on.

2. Document to enable Qt integration with Microsoft's Visual Studio 2008 – This was deemed necessary to compile applications for the windows mobile platform. The essence of this was to allow different uses with different devices to interact irrespective of their mobile devices underlying architecture.

Conclusions and Future Works

This document has presented the overview of a coding project implemented in the Qt codecamp course – CT30A9300. This includes a network based mobile game implemented using Qt widget toolkit and standard C++. The presented QAirHockey requirements, design and implementation suggest a relatively agile cross-platform framework for development and deployment.

To conclude the implementation of a network based mobile air hockey with Qt was completed in relatively short with all the basic features and functionalities. Though not a polished GUI with image or icons for puck, mallets and backgrounds the initial concept was well understood. With the current knowledge incorporating those features will require relative short time thus are left for future version. Incorporated in the future release will be issues addressed in the challenge section and others such as team formations.