Mobile casino gaming

Hazard games for real money goes pocket size

Johan Williams
Linus Nikander

1 INTRODUCTION

Wireless connectivity to the Internet is receiving a lot of media attention at present. This despite the fact that the ways in which one can connect to the Internet wirelessly at the moment are limited to only two solutions, each with limitations that severely restrict their usability. The two available solutions at present are connecting to the net over a dial-up link over a mobile phone, or connecting via a WLAN (Wireless Local Area Network) which in turn is connected to the Internet.

The answer to why media is so interested in wireless connectivity despite the apparent shortage of ways to connect is that several new solutions are about to be released. Some of the solutions will try to remedy the technical restrictions present in current hardware, while others will try to eliminate the need to have separate devices for connectivity and data processing by merging them into one. It has long been said that the next step for the Internet is to make it available everywhere, wireless connectivity will make this possible.

1.1 Problem background

Being a company designing and producing Internet-based gaming solutions, EGET (European Game & Entertainment Technology), has realised that wireless gaming will be an important aspect of the gaming market in the future. Competition within the Internet-based gaming business is already very tough, failing to adopt to new technology as it develops is dangerous for any company that wants to remain competitive.

In light of this EGET decided to set up a project to develop a PDA (Personal Digital Assistant) based client which would give the user a fully functional interface to their Blackjack game, currently playable only with a browser. The current browser-based Blackjack game was developed some time ago, and has been part of the production environment for over two years. The PDA based solution shall be connected to the Internet over a WLAN, and is to be designed in such a way that no changes are needed within the server-side code.

Apart from being a good example of EGETs ability to adopt to new technology, which the company could demonstrate to would-be customers, the project would also give the company valuable knowledge on how to handle future development of games for PDAs or within wireless environments.

1.2 Purpose

The main purpose of this document is to judge if development of a PDA based version of Blackjack is feasible, given the currently available technology. An in depth discussion concerning the technical solution, i.e. how the application was built up, what algorithms have been used and why, is followed by an overview of the problems encountered during development, and the solutions they required. Finally we will analyze the results, and describe the setup, of the testing that was done to verify the applications behaviour.

1.3 Method

The method for the development of the PDA based Blackjack can be divided into a feasibility study, a design phase, a development phase, and finally a testing phase. The method is based mainly on the waterfall development method (Sommerville 2000). The only thing lacking is the operational and maintenance phase. The reason for this is because the application didn't reach a production environment within the scope of the project.

1.3.1 Feasibility study

The first phase of the project was to gather information about PDA application development. Decisions regarding which device and which development tools to use were taken. The goals and specifications from EGET were also discussed and the framework for the project was set up.

1.3.2 Design

With some basic knowledge of PDA development and the specifications for the project given to us by EGET, design work could begin. During this phase the classes and methods needed to complete the project were identified. The design phase also included discussion on how communication with the server should be handled, and how the GUI (Graphical User Interface) for the application should be developed. It was decided that the first version of the application should be entirely text based but that development should be done in such a way that graphics could be easily implemented later on.

1.3.3 Development

In order to get a working version of the application up and running as soon as possible the core objects within the application were the first to be developed. Development was an iterative process where each iteration expanded on the result of the preceding one. We made it a point only to add as much new functionality between iterations as we felt we would be able to test properly before beginning the next iteration. In this way we were able to gradually expand the abilities of the application without ever having to spend a lot of time debugging new functionality because we had added too much at a time.

1.3.4 Test

To find bugs and problems in the application a test plan was designed. As soon as an error was identified the cause of it was sought out and necessary changes to the code were carried out. This process of testing for and correction of errors continued until our test plan could be carried out without errors occurring.

1.4 Scope

The scope of this paper is limited to the design, development and testing of the actual Blackjack application. Many aspects of the already existing serverside solution are mentioned within the paper but are beyond the scope and will not be described further.

Although Windows CE (Compact Edition) is a development platform, meaning that development done for a specific device only has to be recompiled in order to adopt it to other PDAs, the paper will only describe how the application performed on the Compaq iPaq. On paper all PDAs that are Windows CE compatible should exhibit the same behaviour, in practice though this doesn't hold quite true. Writing an application that works on all PDAs that support Windows CE would have required a lot more testing and was therefore deemed beyond the projects scope.

1.5 Disposition

The paper is divided into 6 larger subsections, each of which in turn has several text passages.

The WinOne gaming platform forms the basis of EGETs gaming solution. The WinOne gaming platform includes functionality needed for gaming, handling money transactions, customer registration, and customer care[1]. Chapter 2 describes the background of the EGET WinOne gaming platform. In addition chapter 2 also covers the background of the Pocket PC, the name given to PDAs that operate on the Windows CE operating system. The last part of the chapter covers information needed to understand the key concepts of the Blackjack game, and its current implementation within the WinOne platform.

Chapter 3 deals with the design of the Pocket PC Blackjack. The client / server communication, an overview of the class design, the Blackjack game flow, and the graphics are described. This chapter also discusses security issues and integration with the existing WinOne platform.

Chapter 4 deals with issues regarding the Pocket PC implementation of the game compared to the browser-based version. Other problems, specific to the Pocket PC implementation, are also discussed.

Chapter 5 brings up the testing of the Pocket PC Blackjack. Issues that are answered are how the tests were designed, what was tested, and what the results of the test were. Included in the chapter are also recommendations on how the application could be improved, based upon conclusions drawn from the test result.

Chapter 6 briefly compares the Pocket PC Blackjack with other similar systems i.e. other Blackjack games available on the Pocket PC.

Chapter 7 contains a summary and discussion.

References and used terminology are available in appendices.

[1] (Solutions overview, http://www.eget.fi/solutions.html, 26 Jul 2002)

2 BACKGROUND

To fully understand how the Pocket PC Blackjack has been developed it is important to know the background of the client side consisting of a WLAN equipped Pocket PC as well as the WinOne platform on the server side.

Blackjack is one of the most popular casino games in the world. The rules of the game are described below since some parts of this document can be hard to understand otherwise.

2.1 The evolution of the Pocket PC

The following chapter is based on Michell Rudimans article "The Evolution of the Pocket PC as a Business Tool"[2] .

The original market for PDAs was essentially created by a single company, Palm, with their Palm Pilot line of product. They encouraged users of desktop PCs to bring their agendas, contacts, and tasklists with them by providing them with an easy and automatic way of synchronizing information between the handheld device and their PCs.

While PalmOS, the operating system used on all Palm Pilot devices, has always focused on being as efficient and lightweight as possible, the new wave of PDAs using the Windows CE operating system has provided the users with a host of new possibilities. Not only is the standard Pocket PC 10 times or more faster than the standard Palm Pilot, Windows CE also allows the user to run multiple applications at once, and provides the user with a vibrant display using high resolution and colour.

Whereas Palm Pilots mainly focus on displaying and storing personal information, the standard Pocket PC is also equipped with desktop compatible versions of Word, Excel, and Internet Explorer. Seamless integration and synchronization with your desktop/notebook version of Outlook® means you can take your email, notes, and task lists with you and stay organized and in touch wherever you are.

Pocket PCs also cater for more sophisticated users needs. For instance, the average Pocket PC is equipped with at least 32 Mb (Megabytes) of internal ram. Palm Pilots rarely have more than 8 Mb. This means you can store large complex documents internally. Since an industry standard Compact Flash slot is standard on most of Pocket PCs, you also can add memory and connectivity using Compact Flash peripherals. These peripherals enable you to dial up to the Internet, access a LAN (Local Area Network) using Ethernet, or connect your cell phone for wireless access to your data.

[2] (The Evolution of the Pocket PC as a Business Tool, http://www.microsoft.com/mobile/enterprise/papers/evol.asp, 2001-11-02)

With the growing popularity of handheld computers, new business applications are turning up every day. PocketCAD[3], for example, is fully functional CAD (Computer Aided Design) software that allows you to download CAD drawings from your desktop computer and take them with you. With PocketCAD you can edit, view, and mark up your design drawings wherever you are. AvantGo is another example of a service which, in a short time, has become immensely popular. AvantGo allows you to subscribe to channels that interest you. A channel can be anything from information written and updated by a newspaper or organisation to information updated by a single user. With each synchronization with your computer the information contained within the channels you subscribe to is updated. AvantGo and PocketCAD are only two of a plethora of similar innovative services and applications that are springing up all over the place. As the Pocket PC platform gains even more momentum, and the devices themselves become more and more powerful, so will the services available through them.

2.2 Pocket PC with WLAN Card

The clientside hardware consists of a Compaq iPaq 3360 with 32 Mb of memory and colour display. Additionally the PDA is fitted with an extension slot in the form of a "jacket" into which cards that follow the PC card specification can be inserted. In order to provide the PDA with connectivity a D-link WLAN card is used in the PC card slot. During testing a Nokia 7110 mobile phone was also used to provide the PDA with connectivity. The Nokia can connect with the iPaq through an IR (Infrared) connection.

2.3 Windows CE

The Windows CE platform provides the developer with a stripped down version of the desktop version of Windows API (Application Programming Interface) to work with. APIs have been rewritten, removed and added in order to fit the hardware restrictions imposed by the PDA format[4].

Programming applications that run in Windows CE is done either in Microsoft's eMbedded Visual Basic or eMbedded Visual C++. eMbedded Visual Basic provides a higher level of abstraction from the hardware than eMbedded Visual C++ does. While this can be an advantage if the goal is to accomplish something relatively simple quickly, if there is a need to handle graphics in a more precise manner or a need to encrypt HTTP (HyperText Transfer Protocol) communication then there is no other choice but to use eMbedded Visual C++ since eMbedded Visual Basic doesn't provide either of these features. Both programming languages can be said to be stripped down versions of their Windows equivalents, Visual Basic and Visual C++[5] .

[3] (http://www.pocketcad.com)
[4] (Microsoft Windows CE: An Overview
http://www.wirelessdevnet.com/channels/pda/training/winceoverview.html, 2002-08-05)
[5] (Microsoft eMbedded Visual Tools Product Information
,http://msdn.microsoft.com/vstudio/device/prodinfo.asp, 2002-08-05)

2.4 The evolution of the WinOne Platform

At the time of its launch, in 1999, the WinOne System was the world's first personalized interactive gaming solution specifically engineered for government licensed gaming operators[6]. Initially the system boasted a complete interactive games portfolio, including lottery, casino and sports betting games.
The gaming portfolio has since then been continuously expanded, and now feature more games within each of the key categories.

A key feature within the gaming solution is the close attachment to the BroadVision (BV) platform and the customization features that can be leveraged from it. BroadVision has always been an integral part of the WinOne platform, starting with version 4.1 at the time of its launch all the way to version 6.0 which is the latest release and the release currently implemented within the solution.

2.5 The WinOne Server

The WinOne server solution consists of several different Sun servers, running on the Solaris OS (Operating System), in a networked environment. As the actual configuration of these servers isn't important for this project no further specification is needed.

The front-end for these servers, and the BroadVision platform, consists of Netscape webservers. As far as this project is concerned the software consists of a single interface, the webservers, solely accessible by HTTP communication between the client and the server.

2.6 The rules of Blackjack

As described on the WinOne demo site[7] , Blackjack is played against a dealer (banker). Four decks of cards are shuffled and placed in a card dispenser from which the cards are drawn. Before the cards are dealt, the player places his bet on the table. First the dealer and the player are dealt two cards each. The player may request as many cards as he wishes from the dispenser while the dealer's right to receive further cards is subject to certain rules (see below). The objective is to get as close to 21 as possible. If the player goes over 21, the original stakes will be lost. If the dealer goes over 21, the player who is still in the game wins.
The WinOne platform uses a set of rules that combines the general rules for Blackjack with specific rules, or House Rules.

The value of the cards
Aces are worth 1 or 11 points, face cards (Jack, Queen, King) are worth 10 points and the other cards (2 to 10) are worth their face value.

[6] (Solutions overview, http://www.eget.fi/solutions.html, 26 Jul 2002)
[7] (Blackjack Rules, http://194.112.4.91/demo_gaming1/casino/instructions/info_bot.jsp?BV_SessionID=@@@@1023148488.1029175850@@@@&BV_EngineID=cadcekhjgeglbemicffgcicog.0&category=Spelregler, 12 Aug 2002)

The game

- After the first cards have been dealt, the player chooses to be hit with more cards or to stand. After the player has decided what to do, the dealer takes cards until 17 or more is reached.
- The dealer must always draw another card on 16's and stand on 17's or above.
- If the value of the player's hand is closer to 21 than that of the dealer's, the player wins. The player keeps his stakes and receives a payout corresponding to the size of the stakes (1 to 1). If the player has 21 points after the first two cards are dealt (ace + 10 or a face card), he has Blackjack. In this case the player keeps his stakes and also wins an amount that corresponds to 1.5 times the stakes (3 to 2).
- If the player and the dealer have the same number of points, the game is drawn in a so-called Stand Off. The player keeps his stakes.
- Blackjack always wins over 21 (21 in 3 cards or more).

Split

- If the players first two cards form a pair or have the same denomination, the player can split the cards into two separate hands. When splitting a hand the player must place chips of the same value as the original bet by the new hand.
- Three special rules apply when splitting:
    1. Splitting is only allowed once.
    2. If the player splits a pair of aces, only one additional card is allowed per hand.
    3. A split ace + a face card counts as 21 and not as Blackjack. The same applies for a split face card + an ace.

Double Down

- If the player has 9, 10 or 11 points after the first cards are dealt, the player can double his stakes. Chips of the same value as the original bet are placed next to them on the table. The player can only be hit with one more card. The maximum stakes for the table may be exceeded in this case.

Insurance and Even Money

- Insurance is a separate "game within the game" where the player has the opportunity to win back his original bet when the dealer's first card is an ace and he therefore has the chance of getting Blackjack. Insurance means that the player is betting that the dealer's second card is a face card.
- Insurance is offered if the dealer's first card is an ace. If the player wishes to insure his hand, the player places chips corresponding to half the amount of the original bet in front of the box on the table, on the "Insurance Line." If the dealer gets Blackjack, the original bet is lost but the insurance pays 2 to 1. If the dealer does not get Blackjack, the insurance is lost and the game continues as usual.
- Even Money is offered instead of insurance when the player has Blackjack and the dealer's first card is an ace. If the player chooses to take Even Money it means that the play accepts a payout of 1 to 1 for a Blackjack instead of the normal 3 to 2. Even Money eliminates the risk of a Stand Off in cases where the dealer also has Blackjack.

Rules that apply for the WinOne platform Blackjack:

- The WinOne platform uses four decks of cards.
- The dealer draws two cards for himself, one face up and one face down.
- The cards are shuffled after each round.
- Only one split is allowed.
- All pairs, including a face card + 10 can be split.
- After splitting, one hand is played at a time, starting from the right.
- Split aces can only be hit with one more card.
- Double down is allowed when the value of the original hand dealt is 9, 10 or 11.
- Double down is allowed after splitting, provided that the total value of the two "new" cards is 9, 10 or 11.
- After doubling a "soft" hand (ace + another card) the ace's value is set at 1, a double "soft" 19 + 2 is therefore always worth 11 and not 21.
- It is not permitted to draw cards on 21.
- It is not permitted to count a Blackjack as 11 and then double the stakes.
- Insurance is offered directly on the original hand.
- Even Money is offered directly on the original hand dealt instead of insurance when the player has Blackjack and the dealer has an ace.
- A player may not surrender in return for losing half of his stakes.

3 DESIGN

Designing properly from the start is essential if an application is to be successful without extensive reprogramming. Therefore a lot of time was spent in the design phase of the application. Special care was taken to group functionality into logically coherent groups.

3.1 Purpose of Pocket PC Blackjack

The purpose of the Pocket PC Blackjack is to provide PDA users with the same functionality on their PDAs as is currently available to users via a browser. The Pocket PC version must correctly handle all the states that the game can be put into. The interface must also provide the user with the same options, at the appropriate time, as the browser-based version.

Additionally the application must not require any changes in the serverside code. It must be fully compatible with the current method of communication.

3.2 Client / Server communication

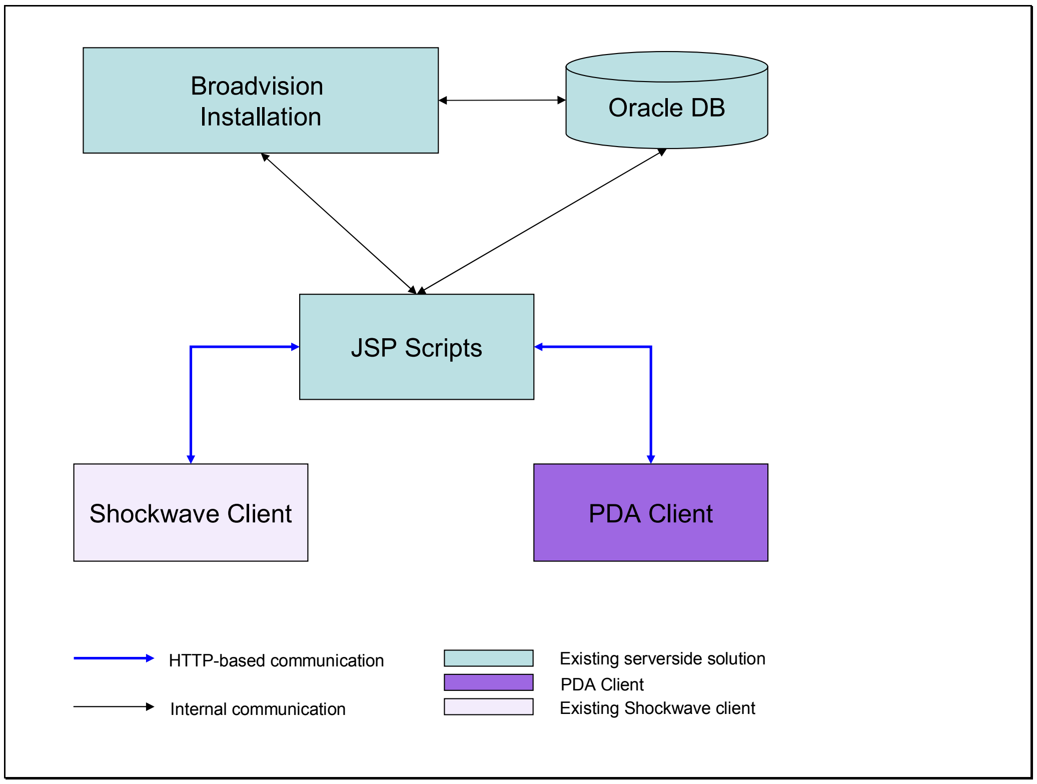


Figure 1 – Client / Server communication

Figure 1 gives an overview of how the different parts of the existing system communicate with each other and of where the PDA developed within this project will fit in.

Essentially a custom HTTP based protocol has been designed to facilitate communication between the Shockwave client and the serverside components. Shockwave is a plug-in technology, used within www-browsers, which vastly increases the interactivity and graphical design possibilities on the client[8]. Using HTML only as the GUI for the game would have made the game a lot less appealing, and would have made development of the clientside a lot more complex. For these reasons Shockwave was chosen as the clientside technology.

The main purpose of the JSP (Java Server Pages) page that lies between the underlying components and the client is to provide a means by which information can be exchanged between the two. JSP pages in turn are executed on webservers that in turn make the appropriate calls to the underlying application and DB (Data Base) servers.

3.3 Class design

All development has been done in C++ using Microsoft eMbedded Visual C++. The skeleton of the project was set up using the built in application wizard. The AppWizard is a code generator that creates a working skeleton of a Windows CE application according to the choices the users makes whilst interacting with the wizard. This provided an initial framework on which the rest of the application was built.

The class hierarchy evolved naturally from the framework generated by the wizard during development of the application. As the Pocket PC BlackJack must communicate with the game server, the first goal was to investigate how to set up a HTTP connection between them. The code needed for this was placed in the connection class, CConnection.

Having successfully established a HTTP connection between the server and the client, the next step in the class development was to try to send and receive HTTP requests between them. This evolved into a HTTP request class.

With the connection set up and the ability to send and receive HTTP packets at hand, the next step was to try to start communicating with the WinOne gaming server using the established protocol.

In order to be allowed to communicate with the WinOne Platform you have to be logged in. The dialogs and logic required for this resulted in the login dialog class.

When you have successfully logged on to the WinOne platform you receive a BroadVision session and engine id. These must be used with every HTTP request to the server in order to authenticate requests during the session.

[8] (Macromedia Shockwave Player – White Paper, http://www.macromedia.com/software/shockwaveplayer/whitepaper, 05 Aug 2002)

During play of the Blackjack game itself there are a number of session specific variables, such as game ids, wallet balance, table stakes, etc. that the game has to keep track of. In order to store all session specific information in a single place, a sessions class was developed.

With all fundamental classes developed we could start concentrating on the actual game functionality and classes. To make them as clean and encapsulated as possible we decided to leave development of graphics until the application was functionally complete. Hence, the first playable version of the Pocket PC BlackJack was entirely text-based.

Although graphics weren't a part of the first few versions of the application, we designed and implemented the application in such a way that graphics could be easily added after initial development was done. The functionality needed to handle graphics was eventually implemented in a class of its own.

Choosing to develop the application in the object-oriented and organized fashion described above resulted in a an application which complies to the original specifications, and which without to much trouble can be understood and further developed by someone else.

The purpose for the development of the Pocket PC BlackJack was to investigate the possibilities of mobile gaming on PDAs. If it is decided that the project is to be used as a product within the WinOne platform there are still quite a few details that have to be taken care of before it can be launched as a commercial product. Among these are:

- **Smaller and better graphics**
  Current graphics are all in bitmap format. This approach was chosen simply because bitmaps, due to the fact that no conversion is needed within the application, are the easiest graphics format to display using Windows CE. The graphics were the least important part of the project and therefore not much time was been spent optimizing this part. The problem with bitmaps is that they are uncompressed and therefore take up a lot of memory, a commodity sorely lacking on most PDAs. If a new version is developed the bitmaps should be replaced with more efficient compressed graphics. More functionality must be added but less disk space will be taken up by the pictures. The actual design of the graphics would also need an overhaul if the product should be made commercial.

- **Added functionality**
  Parts of the functionality found in the Shockwave version of BlackJack were intentionally left out during development, due to their irrelevance within the scope of the project. An example of this is the wallet transaction functionality. When playing on the WinOne gaming platform the money you play with is contained, within the system, in your gaming account. Money is added to your gaming account by transferring it from your bank to the WinOne platform. For financial reasons money used when playing any of the casino games, such as BlackJack, Roulette, and Slotmachine, has to be kept separate from money used elsewhere within the system. In order to achieve this, the notion of a virtual wallet was introduced into the system. When playing one of the casino games,

money spent and won is transferred to and from the wallet. Much like the fact that you have to transfer money from your bank into the system in order to have money to play for, you also have to transfer money from your gaming account into the wallet in order to be able to play one of the casino games. The sum of your money within the system is thus the amount of money within your gaming account added to the amount of money in your wallet. In the shockwave version of BlackJack functionality to transfer money to and from the wallet is part of the application. This functionality hasn't been developed for the PDA based version, due to the fact that the transfers can be done elsewhere within the system. Since the current version has not been exposed to thorough testing it is also possible that more extensive exception handling will have to be implemented in the next version.

- **Sound**
  The current version has no sound effects. This again lies out of the project scope. Much like graphics was easily added when the application had been functionally developed, sounds can easily be added to the current design.

- **Deployment**
  The current version is installed on the Pocket PC by transferring the executable files with eMbedded Visual C++. It is preferable to make a self installing package of the project. Again this was beyond the scope of this project, but should be added if the application is to be made commercial

Due to the object oriented design, and encapsulation of functionality, implementing all additional functionality described above should cause no greater problem. Things left out in this version have been left out due to their irrelevance within the scope of the project, not due to implementation complexity or lack of time.

3.4 Class description

Below follows a brief description of all the classes in the project. A more detailed explanation of the most important functions and variables can be found in the Appendix. Figure 2 gives an overview of all classes within the application.

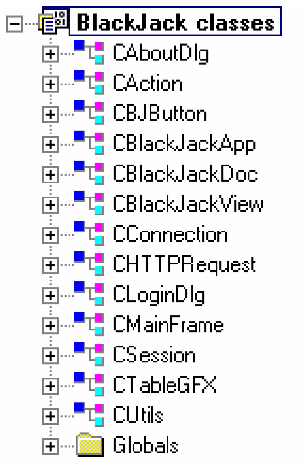


Figure 2 – Overview of the top-level classes within the application.

CAboutDlg
This class is generated when creating the initial framework with the Windows CE MFC AppWizard. It is merely used to display copyright, version, and company information to the user in a small popup window.

CAction
This class interprets the actions taken by the player, both game related actions such as draw, stand, split etc. and the login process. The server only accepts request that are within a valid session. This session is created in the login process. From a given action the corresponding HTTP request can be created that can be sent and understood by the server. The actual sending of the request takes place in the class CHTTPRequest described further below.

CBJButton
Since a lot of buttons are used in the application a button class was needed. The buttons in our applications are sometimes a part of the background, sometimes separate images and sometimes the whole screen (if you want the user to tap the screen to continue for example). They must also be enabled and disabled in different ways and easily moved around the screen. The predefined button class in Windows CE didn't fill our needs for this so we decided to write our own button class. The CBJButton class takes care of all mentioned above as well as the drawing of the actual button. When the pen touches the screen you can call this class to find out weather a specific button was pushed or not.

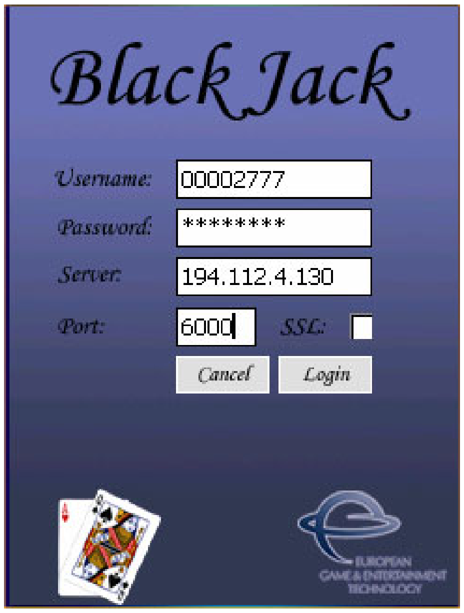
CBlackJackApp
The CBlackJackApp class controls all objects in the project. It is the top-level object class and specifies application behaviour, and handles such tasks as initialization and cleanup.

CBlackJackDoc
The document class contains application specific data. Since the document is only a container for our application there is virtually no application data contained in it. In common windows programs, this is the place where code to handle loading and saving would be placed.

CBlackJackView
The CBlackJackView class is the GUI presented to the user. The view class controls how the user sees the data of the document and interacts with it. All background, card, text and button drawing is initiated from within the class.

CConnection
This class establishes a HTTP session with the server. The class acts as the interface between the application and the server. All requests made to, and responses made by, the server are passed through the class.

CHTTPRequest
Sends and receives HTTP request to a given URL (Uniform Resource Locator). Since real money is involved in the game the application must also handle encrypted requests, and be able to receive encrypted replies, using SSL (Secure Sockets Layer).



Username: 00002777
Password: ********
Server: 194.112.4.130
Port: 6000

Figure 3 – The login dialog

CLoginDlg
This class displays the login dialog box as seen in figure 3. The user specifies username, password, server IP (Internet Protocol) and port to connect to. Optionally the user may force the connection to be encrypted by checking the SSL checkbox.

CMainFrame
Views are displayed inside the document frame windows. The derived main frame-window class specifies the styles and other characteristics of the frame windows.

CSession
The session stores all values needed during gameplay. This is table related values such as min and max stakes for the table, game related values such as bet size, cards, account balance and game id’s and also connection related settings.
Note that the server does never trust the client on any game related data such as cards or account balances. These session values are only used so the client can store and display them in a smart way. The session also avoids unnecessary communication with the server and keeps the protocol slim. The account balance for example only has to be sent to the client when a new game is started or when the player chooses to double his hand or take insurance. In these cases the new account balance is updated on the server and sent to the client so the session value for the account balance can be updated. In this way the session always contains the correct account balance and the client never has to ask the server what it is.

CTableGFX
The CTableGFX class handles all the graphics needed within the Blackjack application. This involves drawing backgrounds, cards and buttons at the correct positions.

CUtils
The CUtils class is a little toolbox class where different useful functions and utilities have been placed. These functions are called from several other classes and had no natural place in any other class and where thus collected in an own class.

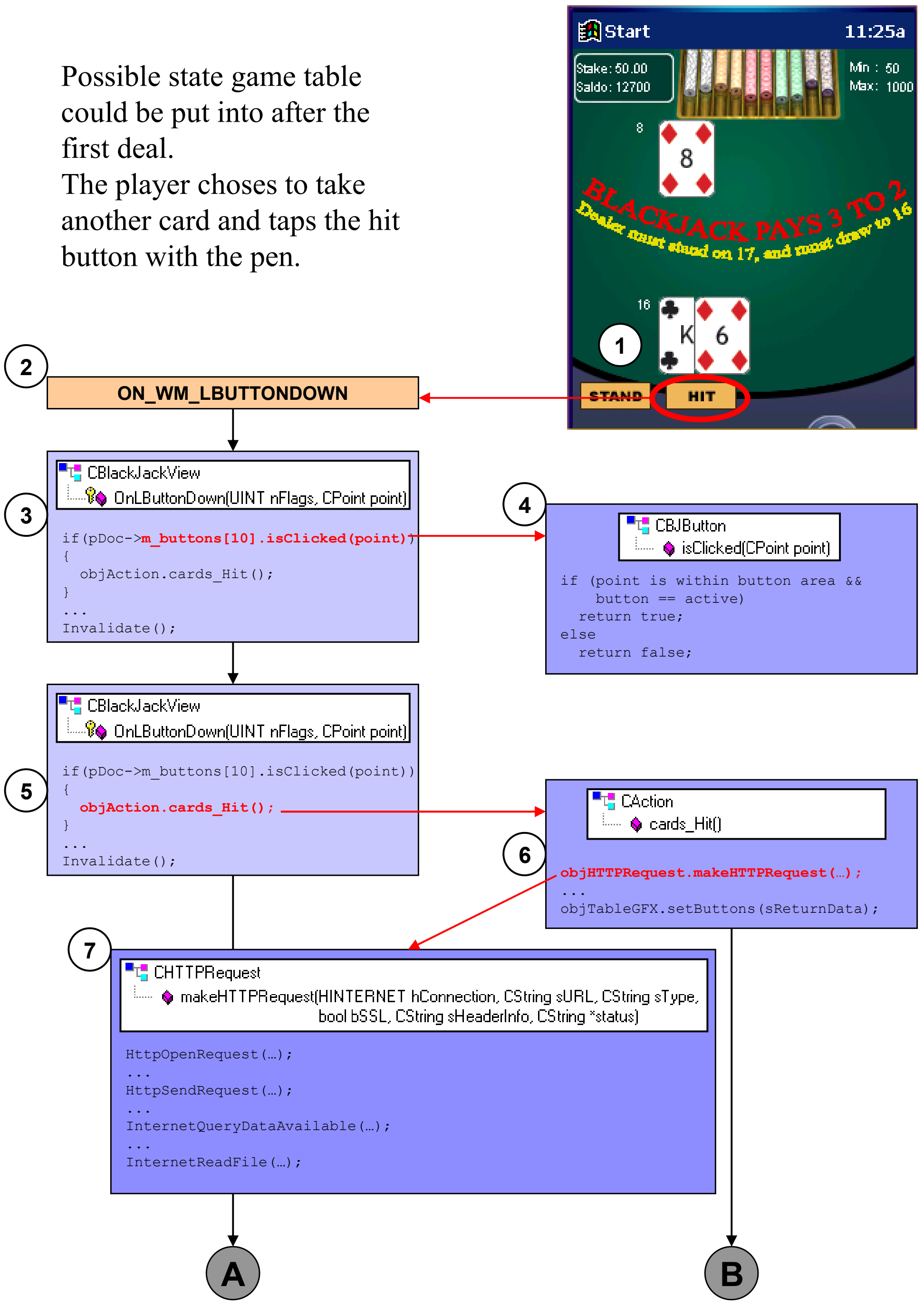
Globals
Most of the classes described above are referenced by many other classes but doesn’t need to be instantiated and created more than once. There is for example no use to have more than one session object.

Global class objects exhibit exactly this behaviour. In practice declaring the class objects as global in the application file makes it available to all classes within the application.

3.5 Class call example

In order to understand and illustrate the class design and in order to visualize the object orientation of the project the following class call example has been included.
The example shows what happens when the user taps the hit button on the game display window. Each step is numbered in figure 4 and described in more detail below.

Possible state game table could be put into after the first deal.
The player choses to take another card and taps the hit button with the pen.



CBlackJackView
OnLButtonDown(UINT nFlags, CPoint point)

```
if(pDoc->m_buttons[10].isClicked(point))
{
  objAction.cards_Hit();
}
...
Invalidate();
```

CBJButton
isClicked(CPoint point)

```
if (point is within button area &&
     button == active)
   return true;
else
   return false;
```

CBlackJackView
OnLButtonDown(UINT nFlags, CPoint point)

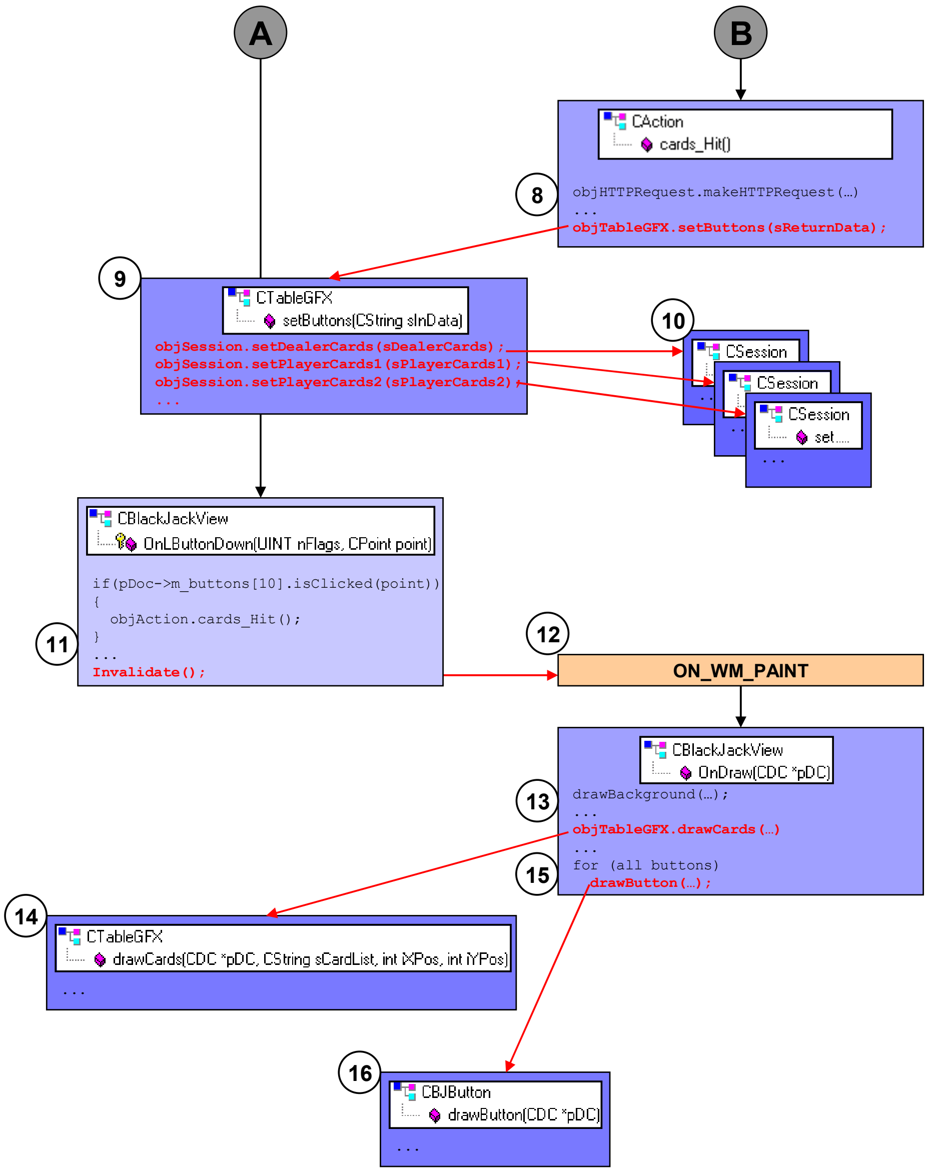
```
if(pDoc->m_buttons[10].isClicked(point))
{
  objAction.cards_Hit();
}
...
Invalidate();
```

CAction
cards_Hit()

```
objHTTPRequest.makeHTTPRequest(…);
...
objTableGFX.setButtons(sReturnData);
```

CHTTPRequest
makeHTTPRequest(HINTERNET hConnection, CString sURL, CString sType, bool bSSL, CString sHeaderInfo, CString *status)

```
HttpOpenRequest(…);
...
HttpSendRequest(…);
...
InternetQueryDataAvailable(…);
...
InternetReadFile(…);
```



```
objHTTPRequest.makeHTTPRequest(…)
...
objTableGFX.setButtons(sReturnData);
```

```
objSession.setDealerCards(sDealerCards);
objSession.setPlayerCards1(sPlayerCards1);
objSession.setPlayerCards2(sPlayerCards2);
...
```

```
if(pDoc->m_buttons[10].isClicked(point))
{
  objAction.cards_Hit();
}
...
Invalidate();
```

```
drawBackground(…);
...
objTableGFX.drawCards(…)
...
for (all buttons)
  drawButton(…);
```

Figure 4 – Class call example when the player wants another card

1. In the GUI, the first deal has been made. The hit and stand buttons are enabled. The user taps the hit button with the pen.
2. Whenever the pen touches the screen the WM_LBUTTONDOWN message is trigged. A message is a way for the operating system to tell an application that something has happened. The message is connected to the function OnLButtonDown()
3. OnLButtonDown() receives the coordinates for a specific point (x, y) where the pen has touched the screen. In order to determine if this point is a button the function calls CButton::isClicked() for all buttons defined in the application.
4. The isClicked function checks if the point (x, y), given as input, lies within the rectangular area in which this button is active. It also checks if this button is enabled at all. If so the function returns true, otherwise it returns false. In the example above isClicked() will return true for the hit button and false for all other buttons.
5. isClicked() returned true and we now know that the hit button is active and was clicked i.e. the player wants another card. This is accomplished by the call to objAction::cards_Hit().
6. cards_Hit() gathers the information needed to make the hit request to the server, and concatenates it into a string which contains valid syntax for the HTTP request. The string is then passed along to makeHTTPRequest in the CHTTPRequest class. After the request has been made makeHTTPRequest returns the response from the server and cards_Hit() can continue processing that information.
7. makeHTTPRequest performs the actual request and returns the servers response. The function itself returns true if the request was successful and false otherwise.
8. After the hit request has been sent to the server the response must be parsed so that the game state on the client can be updated to match the server's action. This is done by calling the setButtons() function in CTableGFX with the servers response. As mentioned before no game logic whatsoever is carried out by the client. The game is played on the server and the client's job is merely to present it to the player.
9. CTableGFX::setButtons() updates the game status according to the response returned by the server. The game status determines which buttons should be shown in the GUI and which should be hidden.
10. After each response a lot of properties, needed in order to maintain consistency between the server and the client, have to be updated. This is done by the set functions in CSession. Examples of such properties are: the dealer and player cards, the player wallet balance, the stake etc.
11. After the above processing has been made it is time to update the GUI so that it corresponds to the new information within the session parameters. To do this Invalidate() is called.
12. Invalidate() triggers the WM_PAINT message which is used to repaint the window.
13. CBlackJackView::OnDraw() is triggered by the WM_PAINT message and draws the current background and the current cards.
14. CTableGFX::drawCards() receives a string of cards to be drawn, for example (4, 64, 108). The cards are matched against a predefined array and the proper graphical representation of that particular card is drawn on screen.
15. After the background and the cards have been drawn the buttons are drawn.

16. CBJButton::drawButton() is used to draw a button to the screen. It is called one time for each button that is to be shown to the user within a particular state.

3.6 Core functionality

Central to the use of the application are two specific parts of the applications functionality, the login- and the gaming process. These two components account for roughly 90% of all time spent designing and developing. Below follows an in-depth explanation of how they both work.

3.6.1 Logging in

Authentication of the users trying to access the system is a central part of the functionality within the WinOne platform. Figure 5 and the text following it explains the different events associated with the authentication. A more detailed and technical explanation is available in the Appendix.

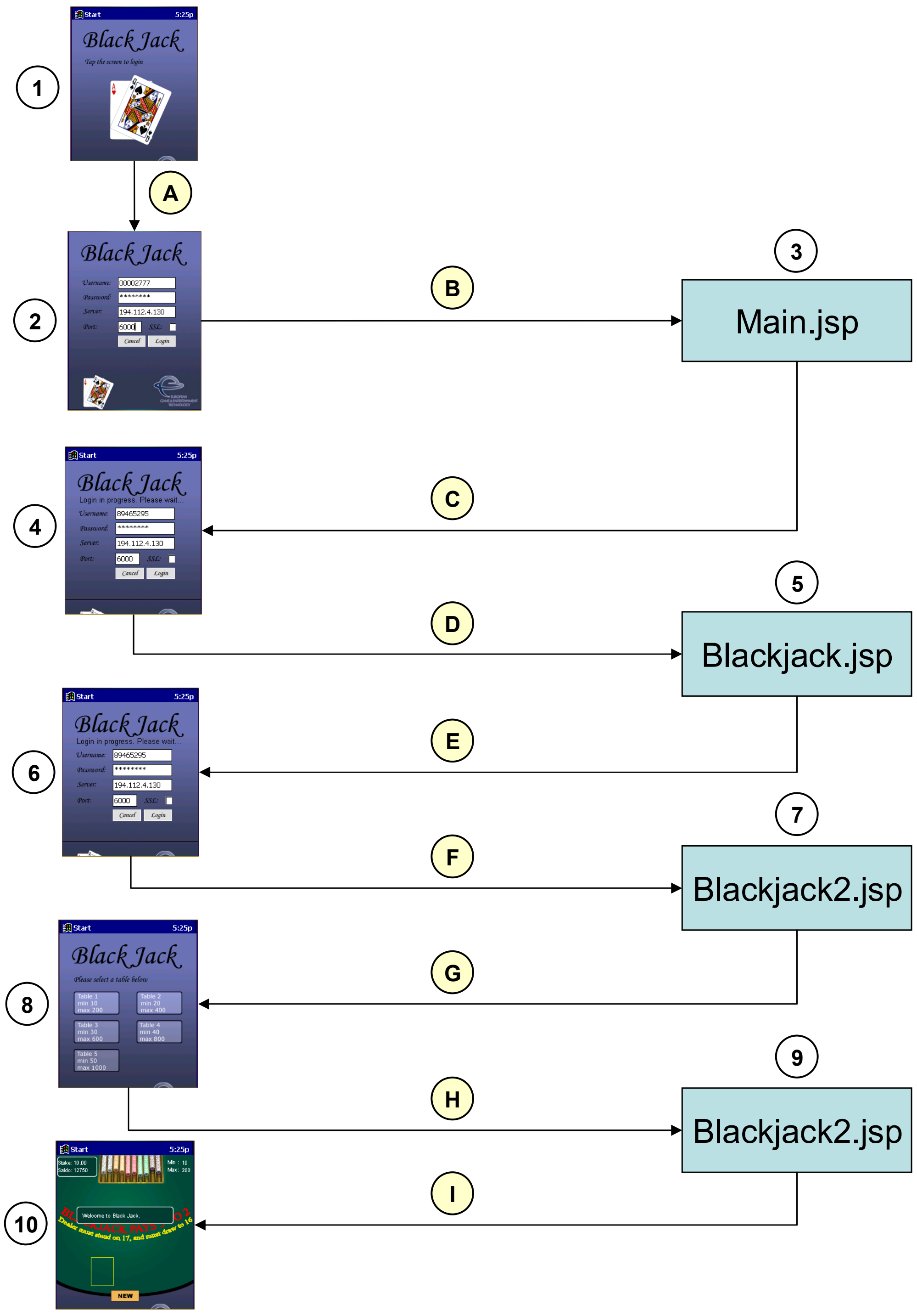


Figure 5- Overview of the requests and responses generated during the login process.

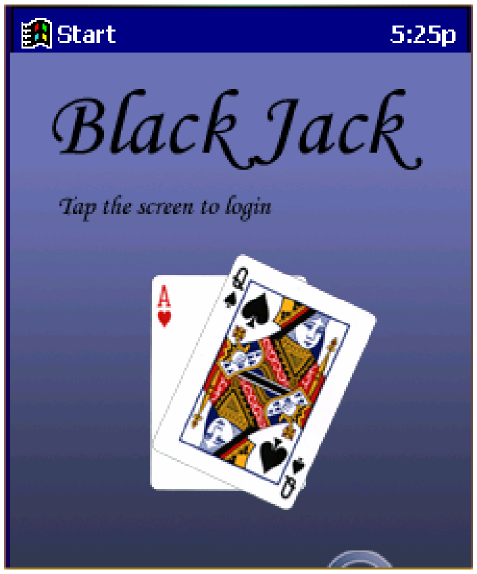
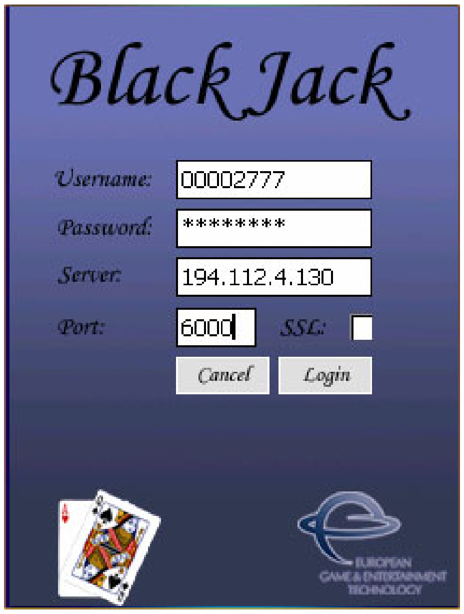


Figure 6 – Splash screen

1. The splash screen shown to the user at start-up of Blackjack.

A. The user taps the login screen.



Username: 00002777

Password: ********

Server: 194.112.4.130

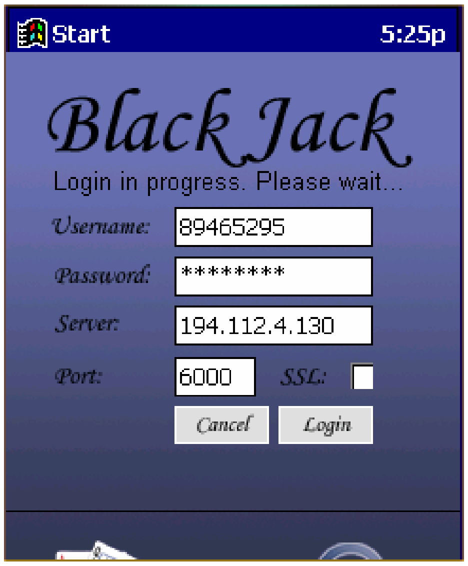
Port: 6000 SSL: ☐

Figure 7 – Login screen

2. The user is shown the login screen. He is asked to fill in the following details:
   - **Username:** The 8 digit username he has been given upon registering.
   - **Password:** The 8 digit password he chose upon registering.
   - **Server:** The IP address of the server he wishes to log on to.
   - **Port:** The port of the server he wishes to log on to.
   - **SSL:** Checked if communication is to be conducted over an SSL-encrypted connection. Left unchecked otherwise.

B. The user taps the login button. The PDA the information to the server for authentication.

3. The authentication is validated.
C. The servers reply is sent to the client.

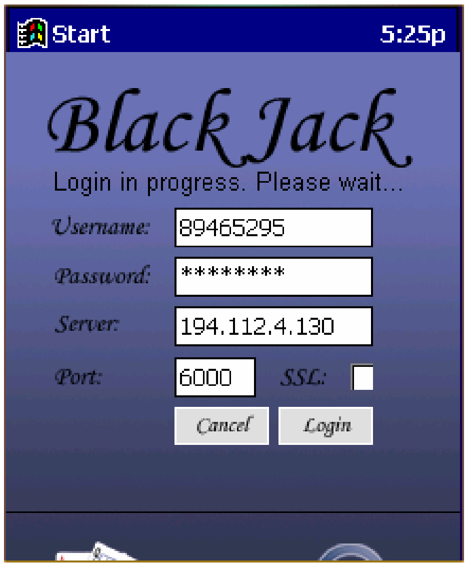


Login in progress. Please wait...

Username: 89465295
Password: ********
Server: 194.112.4.130
Port: 6000 SSL: ☐

Figure 8 – Login screen, communicating with the server

4. The user is shown the "Login in progress" screen indicating that communication is going on with the server he wanted to log on to. The HTML page returned by Main.jsp (C) is parsed. The session unique identifiers, `BV_SessionId` and `BV_EngineId` are extracted and stored in memory on the PDA. These identifiers are needed in all communication within the same session to validate the requests.
D. The PDA initializes an instance of Blackjack by calling BlackJack.jsp.
5. BlackJack.jsp parses the request.
E. The HTML page generated by BlackJack.jsp is sent back to the client.



Login in progress. Please wait...

Username: 89465295
Password: ********
Server: 194.112.4.130
Port: 6000 SSL: ☐

Figure 9 – Login screen, request parsed

6. The user is still shown the "login in progress" screen. The PDA parses the HTML page returned by BlackJack.jsp (E).
F. The instance of Blackjack on the server which is now associated with the session on the PDA is initialized by calling BlackJack2.jsp.
7. BlackJack2.jsp parses the request.
G. The HTML page generated by BlackJack2.jsp is sent back to the client. The reply contains information about the limits for the different tables.

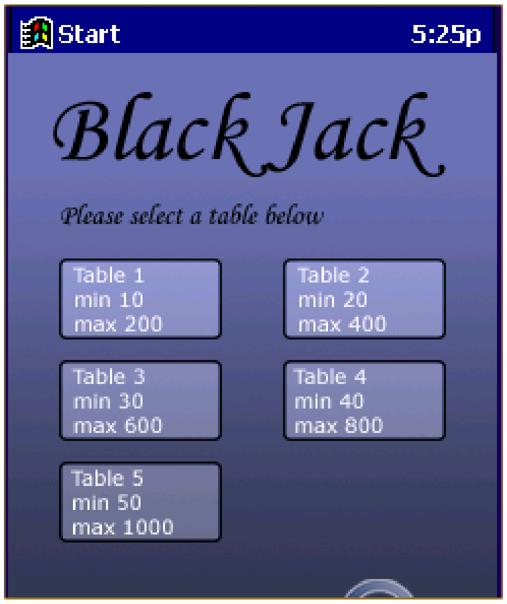


Figure 10 – Table selection screen

8. The user is shown a screen where he is allowed to choose which table he wants to play. The 5 available tables have different minimum and maximum bets, these limits are the values received in the previous request.
H. The user chooses one of the tables by tapping it. The PDA sends the request to BlackJack2.jsp.
9. BlackJack2.jsp parses the request. The reply contains information about the settings for the particular table chosen by the user.
I. The HTML page generated by BlackJack2.jsp is sent back to the client. The reply contains information that initializes the game.

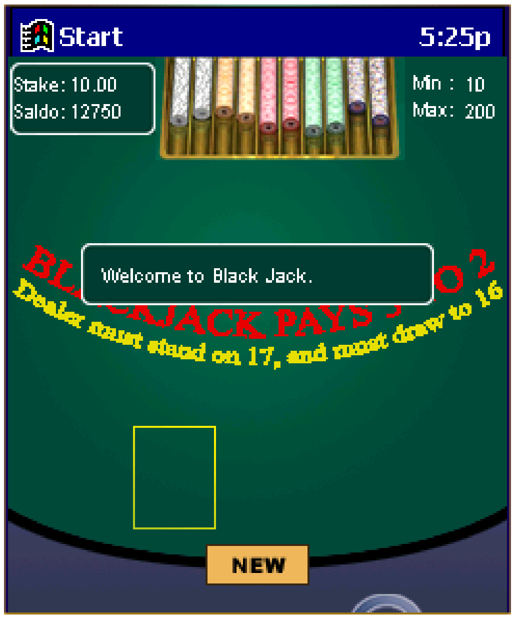


Figure 11 – User logged in successfully

10. The user is shown the “Welcome” interface for Blackjack. At this point all initialization needed by the system has been made, and the user can commence with normal gaming.

3.6.2 Playing Blackjack

The most complex and the largest part of the application is of course the logic required to actually play the game. The underlying diagram and text explains the different events associated with gaming. A more detailed and technical explanation is available in the Appendix**.**

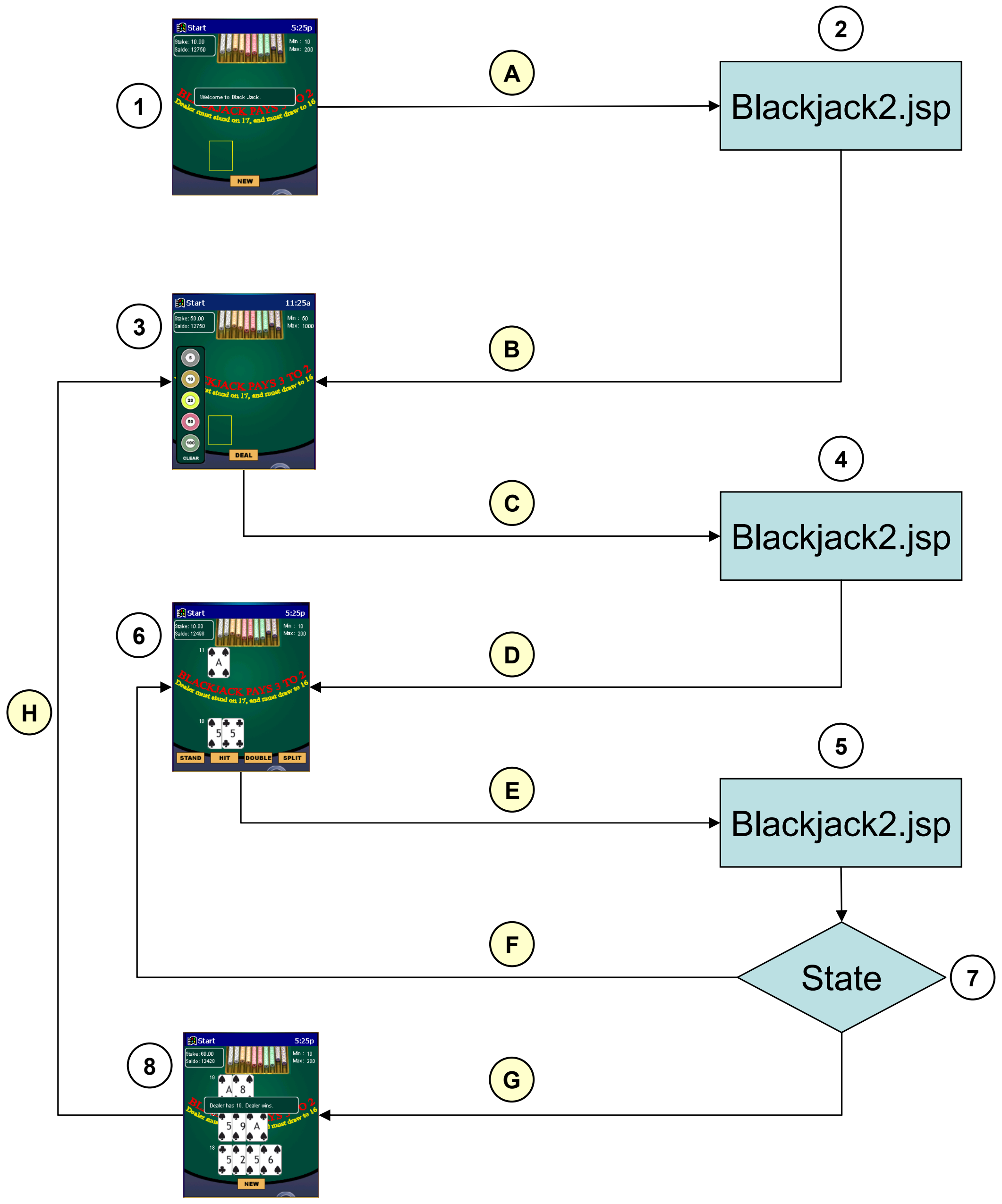


Figure 12 – Overview of the requests and responses that are generated during play of Blackjack.

1. The user is shown the “Welcome” interface for Blackjack. Preceding this is the login procedure described earlier in this document.

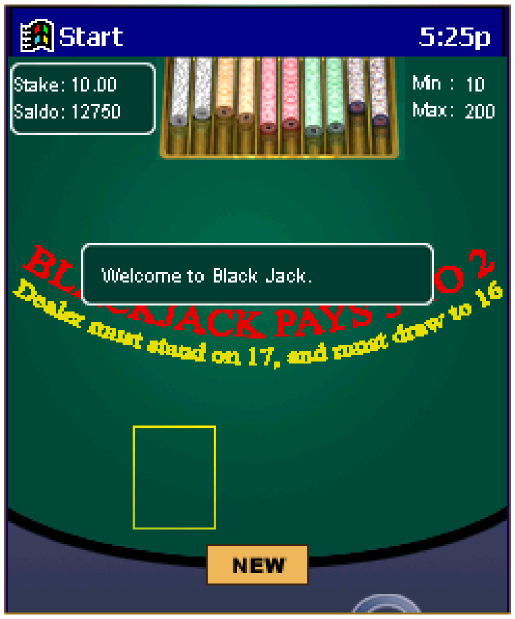


Figure 13 – Start a new game

A. The user chooses to initiate a new game by tapping the “NEW” button.
2. BlackJack2.jsp parses the request and generates the reply.
B. The HTML-page generated by BlackJack2.jsp is sent back to the client.

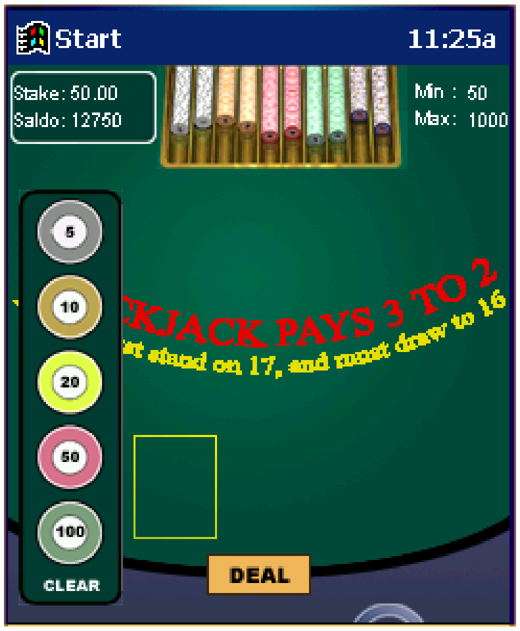


Figure 14 – Bet screen

3. The user is shown the screen where he can decide how much he wants to bet. The default value is the table minimum, shown top right. The player bets by

tapping the markers. His current bet is shown top left. The application only allows him to make bets that are larger or equal to the table minimum, smaller or equal to the table maximum, and smaller or equal to his account balance.

C. The user instructs the server to deal his initial cards by tapping the "DEAL" button. A HTTP POST request is made to BlackJack2.jsp.

4. BlackJack2.jsp parses the request and generates the reply.

D. The HTML-page generated by BlackJack2.jsp is sent back to the client.

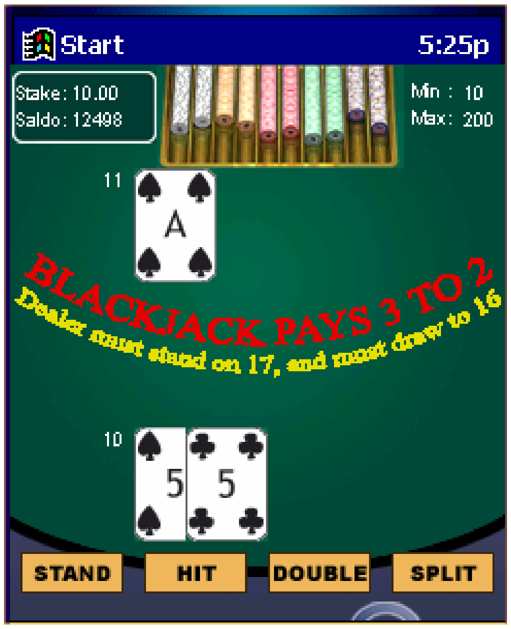


Figure 15 – Game screen

5. The following dialog is shown to the user. Depending on which cards the user has been dealt a different subset of buttons, representing the actions the user can take, is shown. In figure 15 the user has been dealt such a hand that all actions (`Stand, Hit, Double, and Split`) are valid. The user is also graphically shown the cards that have been dealt so far.

E. The user instructs the server to stand, hit, double or split by tapping the one of the buttons he is shown. A HTTP POST request is to BlackJack2.jsp.

6. BlackJack2.jsp parses the request and generates the reply.

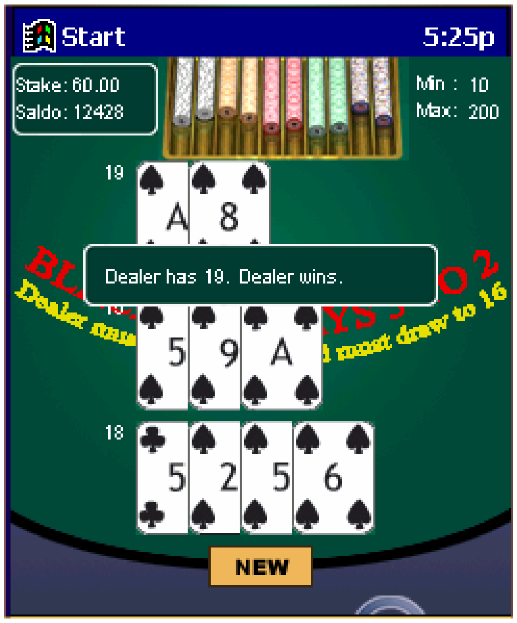
7. As a part of step 5 the server generates a statecode. The statecode tells the application what the result of the users action is. The following states are possible:

0 = The game has not yet been won by any part
1 = Dealer wins with better cards
2 = Dealer wins by Blackjack
3 = Player wins with better cards
4 = The player wins one hand, the dealer the other. After a split
5 = The dealer busts
6 = The player wins with Blackjack
7 = Both hands are a draw after a split
8 = The game is a draw
9 = The player loses one hand and draws the other after a split

10 = The player draws hand one and wins hand two after a split.
11 = The player wins hand one and draws hand two after a split.
12 = Both players have Blackjack, the game is a draw.
26 = The game is closed
50 = Not enough money to cover the action.
55 = Erroneous request or server error.
60 = Too big or too small stake.
65 = The players personal gaming limit is exceeded

F. If the state is 0, i.e. no one has yet won the hand, play continues.

G. If the state-code is something else than 0, i.e. the game has been won by someone, the player is informed of what happened, and is returned to the initial screen where he can continue playing.



Dealer has 19. Dealer wins.

Figure 16 – Game finished

8. The player is shown a screen where he is informed of who won.

H. The player starts a new hand by pressing “New”. The request is the same as step B.

3.7 Graphics

Graphics were designed with two primary goals in mind:

- The graphics should as closely as possible mimic the design and functionality of the original graphics used in the Shockwave version of Blackjack
- The Graphics, especially the cards and the buttons, should be easy to redesign and reposition on the screen.

Graphics on the PDA can be said to be split up into layers. The first layer is the background. On top of this there are layers with buttons and cards. When the screen is rendered on the PDA the first thing that happens is that the background is fetched. After this any buttons and/or cards that should be drawn are rendered on top of the background in memory. When this is done the screen is copied from memory to screen. By using this layering technique it is much easier to manipulate size and position of that which is displayed to the user.

Buttons

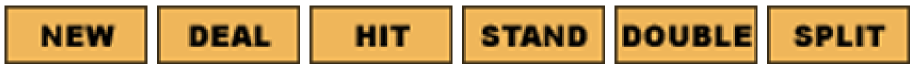


Figure 17 – The buttons

To comply with the above criterion a design was chosen where all graphics that should be easy to redesign were placed in separate files. Thus all buttons are stored in separate resource files within the development environment. The different buttons are shown in figure 17.

If one wants to change the appearance of a specific button, without changing its size or placement, then all one has to do is replace the bmp resource file with a new bmp file containing the new graphics and recompile. If one wants to change the position or size of the button this can also be easily done in the initialization of that particular button.

The cards

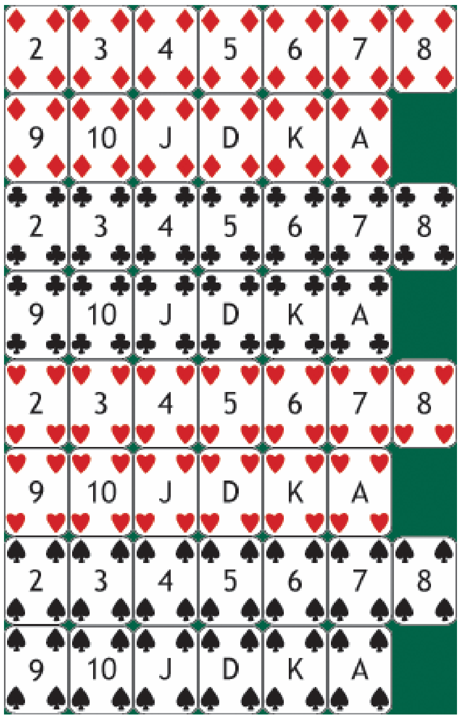


Figure 18 – The cards

As with the buttons the implementation of the cards was such that they would be easy to replace. The cards are stored in a single bmp file shown in figure 18. All cards have the same width and height and thus the same amount of pixels. The fact that the width and height of each card is the same is used when selecting the correct graphics for a particular card. Within the code each card is assigned a set of x and y coordinates stating where in the resource file the card is positioned. For example the three of clubs is assigned position (1,2).

When fetching the graphics for 3 of clubs that information is used to locate the area of pixels within the resource file that contain the image for the three of clubs. The position is calculated as follows:

Left-hand top X coordinate:	1 * CardWidth
Left-hand top Y coordinate:	2 * CardHeight
Right-hand top X coordinate:	1 * CardWidth + CardWidth
Right-hand top Y coordinate:	Same as left-hand top Y coordinate
Left-hand bottom X coordinate:	Same as left-hand top X coordinate
Left-hand bottom Y coordinate:	2 * CardHeight + CardHeight
Right-hand bottom X coordinate:	Same as right-hand top X coordinate
Right-hand bottom Y coordinate:	Same as left-hand bottom Y coordinate.

As long as the layout of the cards within the resource file is kept the same, the graphics can easily be changed to another design. If their sizes are left unchanged the only thing that needs to be done is to replace the bmp resource file and recompile in order for the changes to be implemented. If their sizes are changed in addition to the bmp file change two constants, specifying the width and height of a single card, need to be changed within the code. As this is done in a single place it is very easy to change the look and size of the cards.

The background

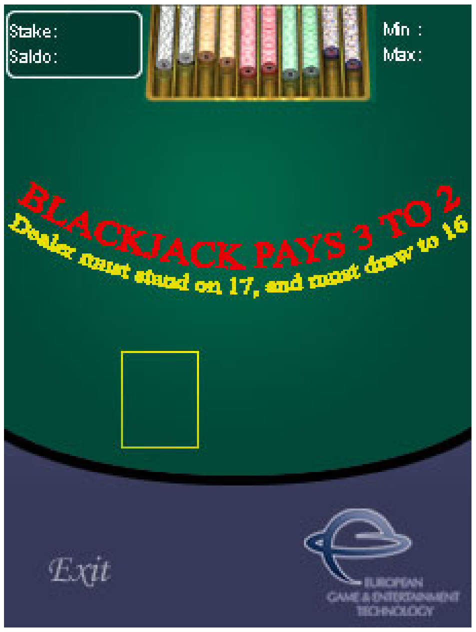

Stake:
Saldo:
Min :
Max:
BLACKJACK PAYS 3 TO 2
Dealer must stand on 17, and must draw to 16
Exit
EUROPEAN GAME & ENTERTAINMENT TECHNOLOGY

Figure 19 – The background

As with all other graphics the background is also a single file, making it easy to replace and redesign. In order to mimic the existing Blackjack as much as possible the background of the PDA version of Blackjack is actually a screen capture from the Shockwave version, slightly manipulated to better fit the PDA constraints.

3.8 Security

For a system which handles transactions where real money is concerned security is naturally of high importance. Within the Blackjack gaming system several layers of security cooperate to provide a secure environment. These can be described as:

- **Client/Server architecture**
  The system has been designed in such a way that the client only acts as an interface which displays actions taken by the underlying servers. This means that no user can ever gain an advantage by gaining an understanding of the HTTP protocol used to communicate between the server and client and then making his own malicious request. The server controls all aspects of which cards are dealt and to whom.

- **The usage off SSL on top of the HTTP connection**
  By using SSL to encrypt all data sent between the client and the server an additional security layer is added which protects the client against packet sniffing. Within the gaming and login processes all information is transferred over an SSL encrypted connection. This means that no information is ever sent in clear text between the client and the server.

- **WEP (Wired Equivalent Privacy)**
  The WEP algorithm is part of the 802.11 WLAN standard is used to encrypt the wireless communication between the PDA and the access point. The WEP algorithm uses 64-bit encryption to add yet another layer of security. Unfortunately due to the way the WEP standard is designed the encryption can quite easily be broken. Although a serious issue in an office environment, this doesn't affect the design of the PDA game that much since the SSL encryption described above will still see to it that captured packets are indecipherable anyway.

On their own, each one of the above security measures can be implemented in order to make an application more secure than it was before. When two or more are used in conjunction with one another within the same application however, security becomes very high and satisfies the high requirements imposed on a system which handles real money. The PDA version of Blackjack relies on all three security measures and can therefore be regarded as transactionally secure.

4 IMPLEMENTATION

The design of the Blackjack application, as described in previous chapters, has been fully implemented. At present the only thing that differs between the Shockwave- and PDA-version is that some parameters that can be set in the Shockwave-version are hardcoded within the PDA version. These parameters are seldom, if at all, changed in the Shockwave version of Blackjack and we therefore felt that development of functionality within the PDA version to handle them would be an unnecessary time investment. The specific parameters that have been hardcoded into the client can be found in the Appendix.

4.1 Experiences from the implementation phase.

The design described above was at first implemented fully on the desktop based emulator. This was done because we knew the connection from the PDA to the LAN via the WLAN card used by the actual device might turn out to be problematic. Because of this we wanted to be sure that once we started testing on the actual device, the application would be fully functional on the emulator ruling it out as a source of problems.

As we had foreseen, problems did arise when we wanted to start testing on the actual device. As it turned out there are very few WLAN cards in public that have Windows CE drivers. The cards that we had bought didn't have Windows CE support at the time, and although we did find articles describing how to get them working anyway, we never attained a connection stable enough to start testing the application. A switch of WLAN cards to ones we knew supported Windows CE solved that problem and finally gave us a satisfactory connection.

Once a stable connection could be maintained over longer periods of time, we started testing the application from the actual device. As it turned out the device ran out of memory when running the application. The cause of this turned out to be the fact that we had forgotten to free memory used when updating the screen. Whenever a change was made to the GUI we first redrew the complete screen, with the changes, to a buffer in memory. This is done in order to reduce flickering which might occur of graphics are drawn directly to screen. The thing we had forgotten to do was to free the memory we drew to. Thus after updating the screen a few times we would run out of memory.

5 TEST

Due to the fact that the application might have to deal with real money at some point in time, rigorous testing had to be made to make sure the application performed as expected. As the serverside part of the solution had already been tested during the design of the shockwave based solution, testing concentrated solely on finding errors made by the client. A main focal point of the testing was to make sure the client displayed the response made by the server correctly. Instances where the client could display erroneous information to the user had to be found and corrected. Testing methodology, results, and evaluation are described in further detail below.

5.1 Brief development testing

During development different parts of the system were tested as the application evolved. This is a natural part of development brought on by the object oriented design approach we took. As each part of the application is forced to start interacting with the other parts of the system, errors were uncovered and remedied. Testing the application as a whole in this way is not acceptable though, therefore a more structured way of testing the application was set up. The details and results of these tests follow.

5.2 Methodical testing

Methodical testing of an application is done at multiple different levels, both within the actual code the application is based upon, as well as on the result the application generates. That means one needs to make sure that the application doesn't just arrive at the correct answer, one must also make sure that the logical path that is followed in order to arrive at the correct answer is correct. Examples of different types of test are:

- **Code review**
  The code, in its entirety, should be reviewed by someone familiar with both the Blackjack game and Visual C++, preferably eMbedded Visual C++. An effort has been made by the programmers to make the high level and inner workings of the code easy to understand. Comments, the conceptual overview of the program, and the module documentation should be given to the person(s) who perform the review. The review should not be carried out by the programmers themselves as it is hard for them to criticize their own code. At the time of writing this test hasn't yet been performed.

- **Functional**
  Functional testing should be carried out to insure that the application can perform all the tasks that it should be able to in a satisfactory manner. Functional testing can be carried out in any environment that behaves, as far as server-responses are concerned, identically with the production environment. In order to make HTTP-requests more easily traceable it is suggested that all tests are performed in a non-SSL environment. Because SSL-doesn't affect the way the application works, it is sufficient to verify SSL by playing the game once in an encrypted environment.

The setup, and result, of the tests that were carried out follow below.

5.2.1 Normal Gameplay

Normal gameplay is constituted of all the states the application must be able to handle during gaming. Typically these tests concentrate on making sure the application displays the correct information at all times, and is able to resume the games in all possible states in case a game is interrupted. Table 1 describes the situations that were tested.

Reference	Description	Suggested test situation
N1	The player shall be able to draw more cards when his cardsum is less than 21	Test behaviour when the player has a cardsum of 2-21.
N2	The player shall be able to split when he has two cards of identical value.	Test behaviour when the player has two cards with values 2-10, or two aces. Test that two different suits can be split.
N3	The player shall be able to split only once.	Deal cards so that the two split piles both contain two identical cards.
N4	After a split an ace + a suited card is counted as 21 and not as Blackjack.	Deal two aces, split deal two cards with a card value of 10. Deal two cards with card value 10, split and then deal aces.
N5	The player shall be dealt only one extra card per split pile if he splits on an ace.	Deal two aces, split. Is checked in N4
N6	The player shall be able to double if his cardsum is 9,10, or 11.	Deal player cards which give him a cardsum of 9, 10, and 11. Deal two 5's to make sure the player can either split or double.
N7	A player shall be allowed to double even if the bet exceeds the maximum for the table	Bet the maximum table amount, deal the user 9,10, and 11.
N8	A player shall be allowed to split even if the bet exceeds the maximum for the table	Bet the maximum table amount, deal the user a split hand (2 cards with the same value).
N9	If the dealer and the player stop on the same cardsum, even money, the money shall be divided evenly between them.	Deal both players cards so that they stop on the same sum. 17,18,19,20, and 21
N10	The player shall be offered the option of insurance when the dealer has an ace.	Deal the dealer an ace and the user something else. Test both when the dealer gets Blackjack and when he doesn't
N11	The player shall be offered the option of even money if he has Blackjack and the dealer has and ace as his first card.	Deal the player Blackjack and the dealer an ace. Test both when the dealer gets Blackjack and when he doesn't

Table 1 – Normal gameplay

5.2.2 Other functions

Other situations are constituted of all the other functions the application must perform, except for gaming. Table 2 describes this functionality and the tests undertaken to make sure the application performed as expected.

Reference	Description	Suggested test situation
O1	The player shall never be able to bet more money than he has in his wallet.	Try betting more than he has in his wallet. Try also to double, split and accept insurance.
O2	The player shall never be able to bet more than the table maximum and less than the table minimum. Exception N6	Try betting more than the max and less than the minimum for all tables.
O3	The user shall not be able to progress past the login screen if he enters an incorrect combination of login and password	Login with incorrect l/p, login with no l/p
O4	The application shall be able to resume a game.	Terminate the game in different states, resume.
O5	The application shall be able to show X cards on the screen at the same time.	

Table 2 – Other functions

5.3 Results

All tests were carried out with a test deck. The test deck can be activated through the monitoring system of the WinOne platform. When the test deck is activated one is able to specify the cards that are to be dealt during the next game. This functionality proved very useful as it allowed us to setup exactly the situations we wanted to verify without much hassle. It could be argued that this type of testing really isn't sufficient to prove that the client can handle all responses made by the server because the randomness present in the real game is removed when one uses the test deck. This point isn't valid though because the server had already been tested separately beforehand. Table 3 and Table 4 list the tests that were performed and the results they generated.

Reference	Execution	Comments
N1	The test deck was set to deal card sums from 2 – 21 to the player.	All tests went ok. Two uncaught states where accidentally discovered and added. 1. Even money (N11) was not implemented 2. Both player gets Blackjack (state 12)
N2	The test deck was set to deal card two cards of same value to the player from (ace, ace) – (king, king)	All test went ok. One uncaught state was accidentally discovered and added. 1. The player wins one hand and looses the other on a split hand. Other win/draw/lose combinations on a split hand was also added

N3	The test deck was set to deal a split hand and also cards of same value to the two new hands.	Test ok.
N4	The test deck was set to deal to (ace, ace) and (10, 10) and give card sum 21 to one or both split hands.	Test ok. Even though the split hand card sum was 21 after 2 cards it paid 1:1 and was not counted as a Blackjack.
N5	The test deck was set to deal 2 aces.	Test ok. Only one card was given to each hand.
N6	The test deck was set to deal card sum of 9 – 11.	Test ok.
N7	The test deck was set to deal a double hand (9,10 or 11) and the table maximum was betted.	Test ok. When doubling the hand the bet exceeded the table maximum.
N8	The test deck was set to deal a split hand and the table maximum was betted.	Test ok. When splitting the hand the bet exceeded the table maximum.
N9	The test deck was set to deal draw hands to the player and the dealer from 17 – 21.	Tests ok.
N10	The test deck was set to deal an ace to the dealer.	Test ok. Insurance handled correctly.
N11	The test deck was set to deal an ace to the dealer and a Blackjack hand to the player.	After adding the even money functionality discovered in N1 all tests where ok.

Table 3 – Results normal gameplay

Reference	Description	Suggested test situation
O1	The wallet balance was set under the maximum table limit.	Test ok. Attempt to bet more then wallet balance impossible.
O2	The wallet balance was set under the maximum table limit.	Tests ok. Trying to bet more/less than the table max/min impossible.
O3	Enter an incorrect username and/or password and try to pass the login screen.	Tests ok.
O4	Close the application in the middle of a deal and then log in again.	Test ok. The game is resumed where it was terminated.
O5	Set the test deck to deal only aces.	Test ok. After 7 dealt cards the cards are drawn outside the visible area of the screen. The card sum is however shown and the game can proceed with no problem.

Table 4 – Results other functions

5.4 Evaluation

The results of the tests proved that the application could handle almost all of the situations it is supposed to. The few problems that did occur were all situations that would almost never occur during gameplay. Nonetheless the application must of course handle those situations as well and the problems were subsequently fixed.

We feel that the methodical approach we took to testing proved itself to be a good way to test the application. Not only did it in fact uncover a few bugs, but it also gave us documented proof that the application can handle the situations it can be put in. This, of course, is invaluable information when considering whether to release the application or not.

6 SIMILAR SYSTEMS

The popularity of the Pocket PC keeps rising and naturally so does the demand for new software. There is software available in almost all kind of categories such as business, personal, education, entertainment etc. Most software, available today, is shareware and costs about $5-25.

6.1 Pocket PC games

Since this application is a Blackjack game connected to a gaming server, over the Internet, we have taken a closer look at similar applications. During our research we didn't find any other games that connected over the Internet in order to play. All Blackjack games we found were a clientside solution and used for entertainment purpose only.

Since the competition amongst gaming companies is fierce, the look and feel of an application is very important. Our application cannot be compared to these ones we found since we have put almost no effort in the graphics making. If an updated version of the application is ever developed, two of the key issues we feel the developers should look more closely at are graphics, and the feel of the game.

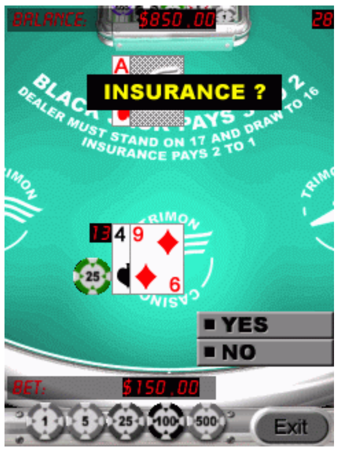


Figure 20 – Pocket Casino by AIDII

Pocket Casino by AIDII[9] comes with 3 different casino games. Besides Blackjack you can also play video poker and a 3 wheel slot machine.
The Blackjack game shown in figure 20 is pretty straight forward. You play against the dealer and the objective is of course to win as much money as possible.

[9] (Products, http://www.aidii.com/home2/product/casino.htm, 14 Aug 2002)

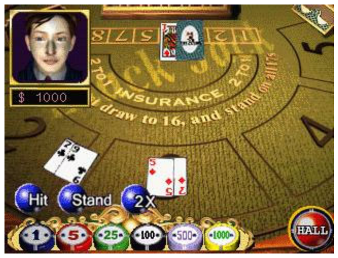


Figure 21 - Blackjack by FLUX

Blackjack by FLUX[10] shown in figure 21 tries to increase the casino feeling by having opponents to play against at the same table. The players faces also appear in the upper left of the screen with different facial expressions depending on how the game goes. FLUX has also tilted the screen 90 degrees to better fit the graphics of the game.

[10] (Games, http://www.flux2game.com/games.htm?code=jack, 14 Aug 2002)

7 CONCLUSION AND DISCUSSION

It doesn't matter how many times you read in a book that software development is no piece of cake, before you've gone through the full circle yourself it doesn't quite sink in. Being put in the position where we, for the first time, were allowed to go through almost the whole cycle of a software development project was good learning experience for the both of us.

We feel that we were able to carry out the project in such a way that it lived up to the specifications made by EGET. In carrying out this task we also succeeded in answering the main question of the project, i.e. if development of a PDA based version of Blackjack was at all possible. We found it not only to be possible to carry out successfully, but to be possible to carry out to the full satisfaction of our project managers.

The only quirk at present is the fact that the result of our efforts isn't used in a production environment yet. This is mainly due to the fact that the market for mobile gaming on PDAs is still in its infancy, and not yet worth investing heavily in.

The development process didn't just result in a finished product, not only have we ourselves gained valuable knowledge where development of mobile applications is concerned, we have also been able to pass this on to our employers. Some of the issues we encountered during the project required a larger effort to solve, than others. Below follows a discussion and our thoughts around these issues. The chapter ends with our thought as to the future of mobile gaming.

During the development of the Pocket PC Blackjack we encountered several problems of varying severity. This is nothing unusual for a medium-sized project like this, and all problems where the solution related to software were eventually solved.

We both have good skills in C++ programming but when you start developing for the Pocket PC you have to change the way you write your code somewhat. First of all, size matters! The hardware used by Pocket PCs severely restricts your options whilst designing and programming. Every byte counts and one must constantly make an effort to make the application as small and efficient as possible. Finding solutions that worked within these restraints was a constant process during the development phase, and ended up taking a lot more time than anticipated.

Another issue which impaired our progress is the fact that the C++ language syntax for Windows CE differs from the ordinary C++ syntax. It is a subset of C++ and a lot of functionality has been left out simply to minimize the size and memory usage of the compiled programmes. Because of this some of the techniques we were used to using couldn't be utilized, and we had to spend time finding alternatives.

We also faced the problem that Pocket PC application development was still in its infancy. In some cases we were trying to write code that did things no one had done before, getting those things to work involved a lot of trial and error. We often got out help from discussion groups on the internet when no book or documentation could help us.

There where also some problems testing the application. eMbedded Visual C++ comes with a Pocket PC emulator that emulates the real device. This emulator was though rather unstable and crashed a lot. Testing the application on the real device was not very trivial either since we had to be connected to the internet via a WLAN. To connect to a WLAN you have to have a WLAN card in your device and an access point (sends and receives the radio signal). We had some problems in finding a card with working Pocket PC drivers and we did not have regular access to the company access point either. WLAN testing days where in other words few.

But as mentioned above there are not only uphill slopes. Since we had been working for EGET for some time before this project we had valuable knowledge and experience. We were already very familiar with the Blackjack server. Once we overcame the threshold of sending a HTTP request and receiving the response from the server we had a functional Blackjack game up and running within a couple of days.

The way we managed the project also contributed in making it so easy to work with. The object orientation and encapsulation ensured that we could work with different parts in the project but still use classes and functionality written by the other developer.

Since the gaming functionality was the same as the existing Shockwave version of Blackjack included in the WinOne platform we could reuse a lot of testing material. Test plans and use cases could be applied directly on this application. The server also features a test card deck that enables the server to deal any cards you specify. This in combination with the test plan ensured that gaming related bugs where discovered and resolved immediately.

The purpose of this application was to show to the customers of EGET that this company is on the forefront of Internet-based gaming systems. To be able to show how a PDA can connect to the WinOne gaming system and then play Blackjack for real money gives a great advantage towards competing gaming system developers. This application might be one of the first in the world to combine mobile casino gaming evolving real money.

But is there a market for mobile gaming? Not yet perhaps. PDAs are not very common and to have them wirelessly connected to the internet is very rare. And just because you own a PDA doesn’t mean you are interested in mobile gaming. But the mobile industry is evolving incredible. Connecting to the internet wirelessly is easier and more common than ever before. Huge effort and money is invested in new and improved mobile networks that strive for connectivity for everybody everywhere. PDAs will most certainly evolve in the same way as the computer i.e. becoming smaller, cheaper and more powerful thus more attractive to the buyers. The demand for different mobile applications will increase along with the growing market. A growing market for PDAs combined with the expanding mobile network is the perfect soil for this application to grow. The seed has been sown and the weather forecast promises sunshine. Lets see if they are right.