

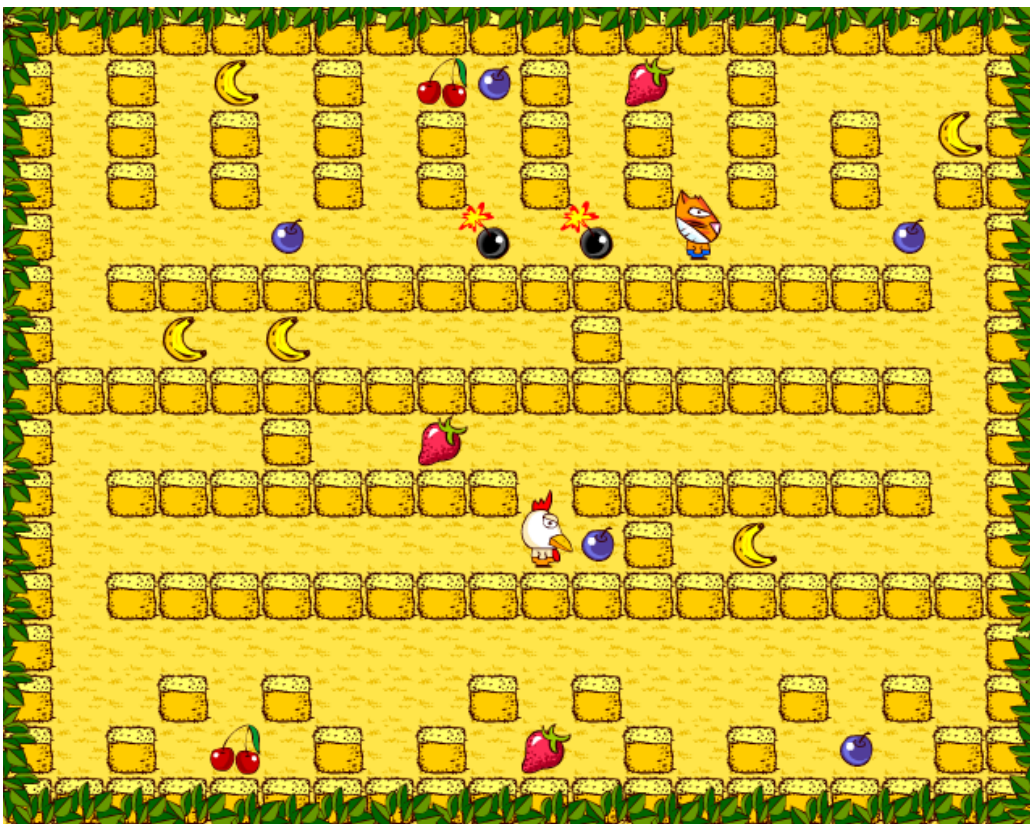


## BattleFarm white paper

January 29<sup>th</sup> 2008, Revision 1

© 2008 SmartFoxServer – [www.smartfoxserver.com](http://www.smartfoxserver.com) – [www.gotoandplay.it](http://www.gotoandplay.it)

**BattleFarm** is a fast-action, real-time multiplayer game built on top of Adobe **Flash 8** and **SmartFoxServer PRO**. The game was created as a case study to highlight the productivity and performance features of SmartFoxServer PRO and its accompanying tools: **SmartFoxBits** and the **BlueBox**.

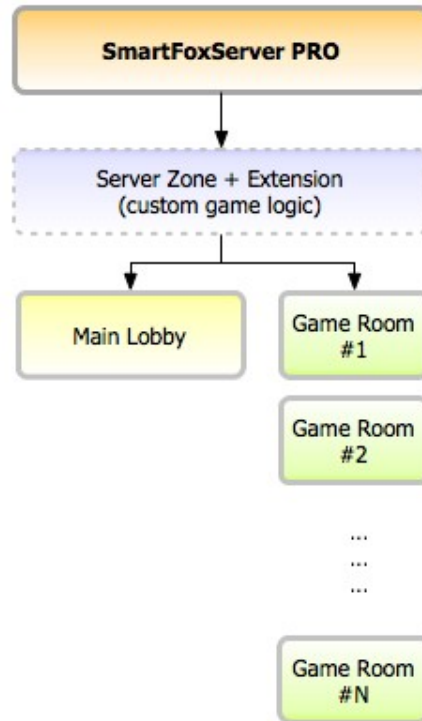


In **BattleFarm** two players compete with each other and against the time to collect the fruit found in the game maze, and using bombs to hit the opponents in order to make them loose what they have collected.

When the timer goes to zero an exit door will randomly appear in the map, and the player with the highest number of collected items will be able to go out and win. In this last phase the opponent can try to block the other player or make him loose his fruits so that he can't leave the map.

## Game Architecture

The architecture of the game is pretty simple: there is one main Lobby room where each users is initially connected and from where he can choose to join or create a new game. A server side extension developed in Java, is responsible for handling the game logic and coordinating the player's events using a highly efficient string-based protocol.



The Lobby system was built in just a few hours with the **SmartFoxBits** components, which allowed to quickly arrange the typical lobby elements such as the room list, user list, public/private chat box, game creation dialogue box etc...

Additionally the **SmartFoxBits** provide a simple and effective way to skin the visual components so that they perfectly integrate with the rest of the GUI design.



## The BlueBox

SmartFoxServer 1.6 introduced the **BlueBox**, a new add-on module designed to allow connections behind firewall and proxies. When the add-on is active the Flash client is transparently redirected to the **BlueBox**, if a direct socket connection is not available.

The **BlueBox** enables players under restricted network configurations to play and enjoy fast multiplayer applications and games with little to no noticeable performance loss. The great news is that all existing SmartFoxServer applications can take advantage of the BlueBox without any code change!

In fact the core of **BattleFarm** was developed way before the add-on was available and we could make the game **BlueBox-aware** by simply recompiling the code with the latest SmartFoxServer client API.

The main advantage of the **BlueBox** compared to other tunnelling solutions is that you can fine tune it to the requirements of your application enabling very good performance even with real time games.

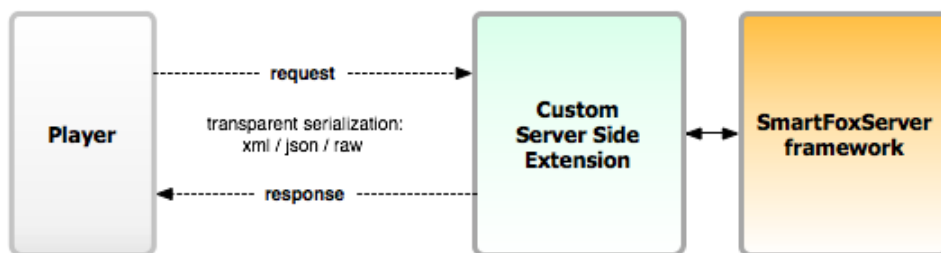
When connected in **BattleFarm** check the bottom-right corner to see what kind of connection was chosen by the client. It could be either **socket** or **bluebox**.



## Server side extension

While most of the Lobby elements are automatically managed by **SmartFoxServer**, the game logic was custom developed using a server side extension plugged at Zone level, which allows to control all the rooms and games running in the application.

Server side extensions provide a simple and very powerful tool for plugging custom application logic that interacts with the core of the server and its framework.



The developer can define any number of custom server messages between client/server and exchange high-level objects which are transparently serialized using various protocols such as xml, json or raw strings.

In **BattleFarm** most of the communication during the game action is performed using the raw-protocol, which allows sending small messages very fast and with optimal bandwidth usage.

## Player synchronization

In order to keep perfect synchronization between players we send minimal amount of updates and we use a mix of client side and server side validation to achieve the best user experience.

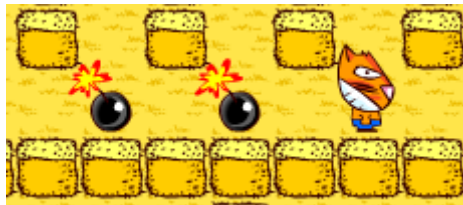
Instead of validating each player move on the server side before actually animating the characters, we perform this check from the client side and notify the server about it. This way the game always feels smooth and highly responsive, regardless of the network lag.

Collision checking with the collectible items is done on the server-side because we need to make sure that only one player grabs an item from the map. Similarly the bomb explosion is partially done on the server side: when a user drops a bomb it will notify the server which will in turn start a countdown to the explosion. When the timer expires all clients are informed that the explosion should take place and the animation and relative collision checks are performed on the client.

One of the most important aspects of the Flash side of the game is using **time-based animations**, which ensure consistent execution of all game animations regardless of the computer CPU speed, memory, video card etc...

In fact one of the difficulties of client synchronization is represented by the potential differences in rendering performance of the Flash Player. In other words a player running an old PentiumIII with 128Mb of RAM would experience slower animation rendering of another player running a shiny new multi-core machine with several gigabytes of RAM.

By ensuring that each animation takes the exact same time on any machine we eliminate the risk of additional client lag and we just need to deal with the possible delays caused by the network lag.



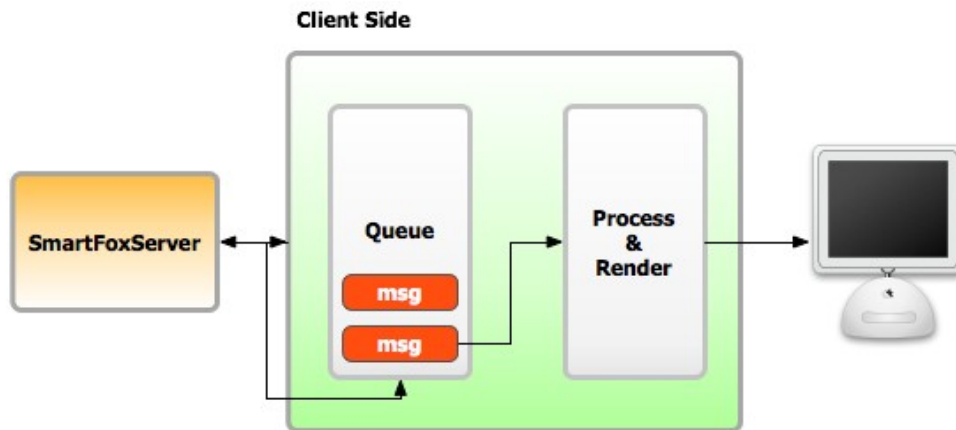
## Handling slower connections

In order to keep up with possible connection slow downs we maintain on the client a queue of moves sent by the server and we execute them serially by grabbing the first item from the queue and processing it before moving on to next one.

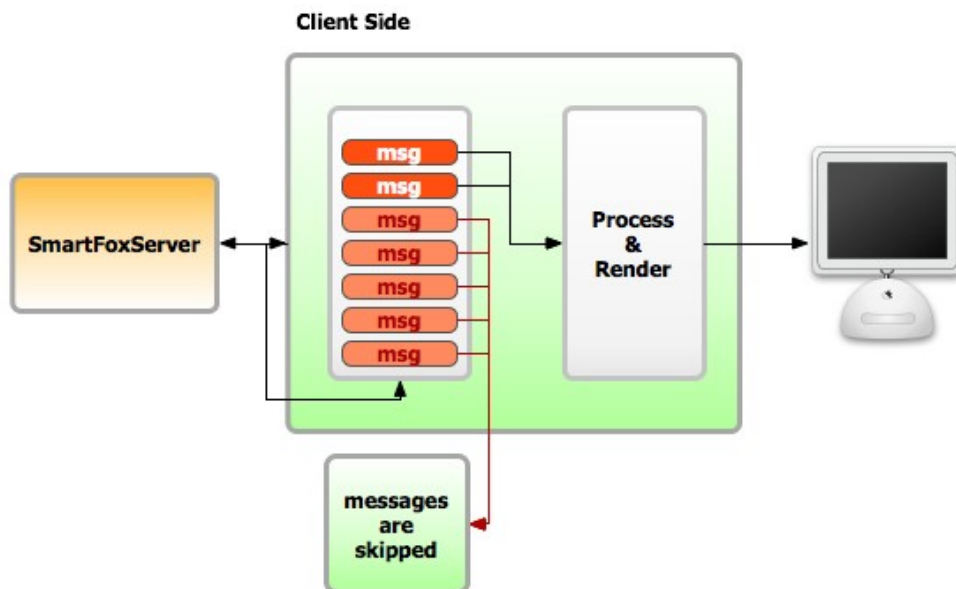
If the client connection becomes congested for a while, many of the incoming messages won't be received for some time until the network becomes available again: at that point all the messages will suddenly arrive and populate the client queue.

If those messages are too many the client will automatically skip them in order to keep up with the latest character positions, and the player will probably notice a certain amount of frame-skip in the opponent animations.

The following diagram shows a "regular" client queue state:



As new messages are enqueued in the client they are processed and rendered. In case the client can't keep up with the server message speed rate (e.g. after a network congestion) older messages are discarded to keep up with the latest game state.



If you want to learn more about this technique you will find a detailed tutorial with source code in the examples provided with **SmartFoxServer PRO** (chapter 8.6, "Real Time Maze")

[http://www.smartfoxserver.com/docs/docPages/tutorials\\_pro/06\\_realtimeMaze/index.htm](http://www.smartfoxserver.com/docs/docPages/tutorials_pro/06_realtimeMaze/index.htm)

Further informations about the **SmartFoxBits** components and the **BlueBox** add-on are found here:

<http://www.smartfoxserver.com/bits/>

<http://www.smartfoxserver.com/docs/docPages/blueBox/overview.html>

Visit also:

<http://www.smartfoxserver.com/>