Non-local Boxes: Theory & Implementation in Minecraft

BACHELOR THESIS

by

Timo Yannick Simnacher

Faculty of Mathematics, Computer Science, and Natural Sciences
RWTH Aachen University

April 4, 2016

Supervised by
Prof. Dr. B. M. Terhal
Institute for Quantum Information
and
Prof. Dr. J.-P. Katoen
Chair for Software Modeling and Verification
Acknowledgements

First of all, I would like to express my gratitude to professor Barbara Terhal for supervising the thesis. She has been extraordinarily tolerant and supportive. Without her guidance and persistent help this thesis would not have been possible.

Special thanks also to professor Joost-Pieter Katoen for providing Erstbetreuung formally. His confidence in my work and professor Terhal’s guidance proved most helpful to me.

Last but not least, I have greatly benefited from my parents’ and my brother’s full support.
# Contents

<table>
<thead>
<tr>
<th>List of Figures</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
<td>ix</td>
</tr>
<tr>
<td>Outline</td>
<td>x</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Superdense Coding ................................................. 4
1.2 Bell Inequality .................................................. 5

## 2 Non-local Boxes

2.1 Non-signaling Condition ......................................... 12
2.2 CHSH-Game ......................................................... 12

## 3 Consequences of Superstrong Nonlocality

3.1 Communication Complexity ....................................... 15
3.2 Secure Function Evaluation .................................... 17
3.3 Oblivious Transfer ............................................... 18
3.4 Information Causality ............................................ 19

## 4 Implementation in Minecraft

4.1 QCraft .............................................................. 22
  4.1.1 Observer Dependent Block .................................. 22
  4.1.2 Quantum Block ................................................ 23
  4.1.3 Entangled Block .............................................. 24
  4.1.4 Quantum Computer .......................................... 25
  4.1.5 Additional Features ........................................ 27
  4.2 PR-box Modification ............................................ 27
    4.2.1 Spatial Partition ......................................... 31
    4.2.2 Re-use ..................................................... 32
    4.2.3 Timed Synchronization .................................... 34
    4.2.4 Technical Implementation ................................ 35

## 5 Summary

## Bibliography

## A PR-box Modification Code
List of Figures

1.1 Superdense coding between Alice and Bob ................................................. 4
1.2 CHSH-inequality setup (classical) .............................................................. 6
1.3 CHSH-inequality setup (quantum) ............................................................... 6
2.1 Setup of a non-local box .............................................................................. 9
2.2 Classical setup .......................................................................................... 10
2.3 Setup of a quantum non-local box ............................................................. 10
2.4 Probability distribution of the quantum non-local box using the singlet state .. 11
2.5 Setup of a PR-box ...................................................................................... 11
2.6 Setup of the CHSH-game ........................................................................... 13
3.1 One-out-of-two oblivious transfer ............................................................... 18
4.1 A typical Minecraft scenery in third person perspective .............................. 21
4.2 The crafting recipe of the essence of observation out of four quantum dust (a) and an observer dependent block using the essence of observation (b) .......... 22
4.3 The crafting recipe of the essence of superposition out of four quantum dust (a) and a quantum block using the essence of observation (b) ................ 23
4.4 The crafting recipe of the essence of entanglement out of four quantum dust and one essence of superposition (a), two entangled blocks using the essence of entanglement and two identical observer dependent blocks or quantum blocks (b), and adding another identical block to a group of entangled blocks, one of them representing the group in the recipe (c) ......................................................................................... 24
4.5 A matrix that has to be built to quantize and teleport blocks in qCraft ...... 25
4.6 The crafting recipe of the quantum computer out of seven iron ingot, one quantum dust, and one glass pane (a) and two entangled quantum computers using the essence of entanglement and two unentangled quantum computers (b) ................. 25
4.7 A portal that enables player to travel vast distances within the blink of an eye . 26
4.8 The crafting recipe of quantum goggles using one quantum dust and two glass panes ......................................................................................... 27
4.9 The crafting recipe of anti-observation goggles using one essence of observation and two glass panes (a) and automated observers using seven stones, one essence of observation, and one redstone dust (b) ................................................................. 27
4.10 A lever and a conduit made of redstone dust (a) which is powered in (b) .... 28
4.11 Two NOT circuits built out of an ordinary block and a redstone torch which is powered (top) or non-powered (bottom) (a), an OR circuit (b), and an AND circuit .......... 28
4.12 The crafting recipe of a PR-box block out of two quantum computers (a) and the entanglement creation using the essence of entanglement (b) ................................................................. 29
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.13</td>
<td>A PR-box before (a) and after (b) energizing using levers to power the inputs. When energized, the block’s color changes from blue to green.</td>
</tr>
<tr>
<td>4.14</td>
<td>Examining the correlation group of a PR-box.</td>
</tr>
<tr>
<td>4.15</td>
<td>Addition modulo 2 using redstone primitives.</td>
</tr>
<tr>
<td>4.16</td>
<td>Computing the inner product $x_1 \cdot y_1 \oplus x_2 \cdot y_2$ distributively using one communicated bit ($x_1$ at the top and $x_2$, $y_2$ at the bottom levers; output at the very top). Correlated PR-blocks are located opposite each other.</td>
</tr>
<tr>
<td>4.17</td>
<td>Computing the equality function $EQ(x_1, x_2, y_1, y_2)$ distributively and securely using one communicated bit (top left: $y_1$, bottom left: $y_2$, top right: $x_2$, bottom right: $x_1$; output at the very bottom). Correlated PR-blocks are located opposite each other.</td>
</tr>
<tr>
<td>4.18</td>
<td>Oblivious transfer (at the very top from left to right: $x_1$ and $x_0$, in the middle: $c$ and at the very right $x_c$).</td>
</tr>
<tr>
<td>4.19</td>
<td>Two correlated PR-blocks with inputs 1 and 1 and outputs 1 and 0 (a). Then, Alice changes her input to 0 and the outputs adapt (b). They could get both 1 or both 0 each with probability $\frac{1}{2}$. Now Bob changes his input to 0 and the outputs adapt (c). Again, they could also get both 1.</td>
</tr>
<tr>
<td>4.20</td>
<td>Noisy channel using repeated energizing.</td>
</tr>
<tr>
<td>4.21</td>
<td>The crafting recipe of a timed PR-box using a clock and two ordinary PR-boxes (a) and its appearance in the Minecraft world (b).</td>
</tr>
</tbody>
</table>
Notation

1 - identity operator

$\alpha, \beta, \ldots$ - complex numbers

$i$ - imaginary unit

$\alpha^*$ - complex conjugate of $\alpha$

$|\psi\rangle$ - vector of a Hilbert space

$\langle \psi | \psi' \rangle$ - inner product in Dirac-notation

$|\psi\rangle \otimes |\psi'\rangle$ - tensor product

$U^\dagger$ - adjoint operator of $U$

$\sigma_0 = 1, \sigma_1 = X, \sigma_2 = Y, \sigma_3 = Z$ - Pauli operators

$\langle \psi | A | \psi \rangle$ - expectation value of observable $A$

$\text{tr}(\cdot)$ - trace of an operator

$U = \begin{pmatrix} & & \vdots & \\ & & & \vdots \\ \vdots & \ddots & & \end{pmatrix}$ - matrix representation of an operator

$[Q, R]$ - commutator of two operators

$P(a|x)$ - conditional probability of $a$ given $x$

$\bar{x}$ - negation of the bit $x$

$x \oplus y$ - addition modulo 2
Outline

The thesis is divided into two parts. Part I provides an introduction to quantum computation and Bell inequalities in the first chapter. In the second chapter, these are applied in an information theoretical setting to introduce the PR-box as a device which maximally violates the CHSH-inequality. In chapter 3, it is shown that for certain tasks, non-local boxes with super-quantum correlations allow superior performance.

Part II describes the implementation in Minecraft. The qCraft modification serves as a starting point for the implementation of PR-boxes. The different elements in qCraft and their analogues in quantum mechanics are examined and compared. Different approaches to the PR-box modification are presented and the final implementation is described in detail.
Chapter 1

Introduction

In this first chapter, the introduction of so-called non-local boxes is motivated. The idea of these boxes has arisen from observations in quantum mechanics. A comprehensive introduction to quantum computation and quantum information can be found in [14]. The following considerations are based on the descriptions from this source.

In contrary to a classical bit, a qubit can be in a superposition \( |\psi\rangle = \alpha |0\rangle + \beta |1\rangle \) with \( \alpha, \beta \in \mathbb{C} \) and \( |\alpha|^2 + |\beta|^2 = 1 \) where |0\rangle and |1\rangle are identified with the states 0 and 1 of a classical bit. Therefore, the state space of a classical bit is \( \{0, 1\} \) whereas the state space of a qubit is \( \mathbb{C}^2 \). This space is a Hilbert space with inner product \( \langle \psi | \psi' \rangle \) with \( \langle i | j \rangle = \delta_{ij} \) for \( i, j \in \{0, 1\} \) and \( \langle \psi | = |\psi\rangle^\dagger = \alpha^* \langle 0 | + \beta^* \langle 1 | \), the dual state of \( |\psi\rangle \) with \( \alpha^*, \beta^* \) the complex conjugates of \( \alpha, \beta \). Thus, any state is represented by a unit vector with respect to this inner product because of the restriction \( |\alpha|^2 + |\beta|^2 = 1 \).

\( \{ |0\rangle, |1\rangle \} \) is called the computational basis and we can use this basis to express a state \( |\psi\rangle = \alpha |0\rangle + \beta |1\rangle \) in vector form \( |\psi\rangle \approx \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \). Quantum states can be manipulated by linear unitary operators \( U \), i.e. \( UU^\dagger = U^\dagger U = 1 \). Important examples for such unitary operators are the Pauli operators or Pauli matrices:

\[
X \doteq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y \doteq \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z \doteq \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
\]

In the following, the terms operator and matrix are used interchangeably. The Pauli matrix \( X \) is also known as the bit flip operator because of \( X |0\rangle = |1\rangle \) and \( X |1\rangle = |0\rangle \).

If we want to know whether a classical bit is in the state 0 or 1, we will simply "look" at it. Measuring a qubit \( |\psi\rangle \) is more involved. One postulate of quantum mechanics says that a quantum measurement is done by a set \( \{M_m\} \) of linear measurement operators that satisfy the completeness equation \( \sum_m M_m^\dagger M_m = 1 \) where \( M_m^\dagger \) is the unique operator that satisfies \( \langle \psi' | M_m |\psi\rangle = \langle \psi' | M_m^\dagger |\psi\rangle^* \) for all \( |\psi\rangle, |\psi'\rangle \) in the corresponding Hilbert space where \( M_m^\dagger \) operates on the state written on the left-hand side. The measurement outcome is \( m \) with probability \( p(m) = \langle \psi | M_m^\dagger M_m |\psi\rangle \). After the measurement, the measured state \( |\psi\rangle \) has become the
state \( \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m M_m |\psi\rangle}} \) according to the measurement outcome.

An important special case of such measurements are projective measurements. A qubit is said to be measured in an orthonormal basis \( \{ |\phi_1\rangle, |\phi_2\rangle \} \) of \( \mathbb{C}^2 \), i.e. \( \langle \phi_i | \phi_j \rangle = \delta_{ij} \). Using this property, the state \( |\psi\rangle \) can be written as \( |\psi\rangle = \gamma |\phi_1\rangle + \delta |\phi_2\rangle \) with \( |\gamma|^2 + |\delta|^2 = 1 \). The measurement in this basis yields \( \phi_1 \) with probability \( |\gamma|^2 \) and \( \phi_2 \) with probability \( |\delta|^2 \). After the measurement, the state \( |\psi\rangle \) has become either \( |\phi_1\rangle \) or \( |\phi_2\rangle \) according to the measurement outcome. Thus, the measured state is projected on the base vectors. The measurement operators are \( M_{\phi_1} = |\phi_1\rangle \langle \phi_1 | \) and \( M_{\phi_2} = |\phi_2\rangle \langle \phi_2 | \). For a vector \( |\psi\rangle = \gamma |\phi_1\rangle + \delta |\phi_2\rangle \) it follows that \( \sum_m M_m |\psi\rangle = \gamma (|\phi_1\rangle \langle \phi_1 | + |\phi_2\rangle \langle \phi_2 |) + \delta (|\phi_1\rangle \langle \phi_2 | + |\phi_2\rangle \langle \phi_1 |) = \gamma |\phi_1\rangle + \delta |\phi_2\rangle = |\psi\rangle \). Thus, the completeness equation is satisfied and the projective measurement is a special case of quantum measurements described by the quantum mechanics’ postulate.

Example 1.0.1. The Pauli operators can be used to specify projective measurements. The orthonormal basis is determined by the eigenvectors and the possible measurement outcomes are the corresponding eigenvalues. Such operators are called observables.

\[
X = |+\rangle \langle +| - |−\rangle \langle −| = \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) - \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right),
\]
\[
Y = \left( \frac{|0\rangle + i |1\rangle}{\sqrt{2}} \right) \left( \frac{|0\rangle + i |1\rangle}{\sqrt{2}} \right) - \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right),
\]
\[
Z = |0\rangle \langle 0| - |1\rangle \langle 1|.
\]

Every Pauli matrix has eigenvalues \( \pm 1 \). We can also have a look at the expectation value \( E(\cdot) \) when measuring a state \( |\psi\rangle \) which is specified by the operator \( A \) with eigenvalues \( a_0, a_1, \ldots, a_n \in \mathbb{R} \) and orthonormal eigenvectors \( |v_0\rangle, |v_1\rangle, \ldots, |v_n\rangle \).

\[
E(A) = \sum_{i=0}^{n} a_i \cdot p(a_i) = \langle \psi| \left( \sum_{i=0}^{n} a_i |v_i\rangle \langle v_i | \right) |\psi\rangle = \langle \psi| A |\psi\rangle
\]

which is the Dirac-notation of the expectation value. It is sometimes shortened to \( \langle A \rangle \) if the measured state is clear from context. For instance, measuring the state \( |−\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \) with the Pauli matrices yields
described by the density operator

A quantum system that is in each of the states

Definition 1.0.2.

\begin{align*}
\langle X \rangle &= \langle -|+\rangle \langle +| -\rangle - \langle -| -\rangle \langle -| -\rangle = -1, \\
\langle Y \rangle &= \frac{1}{\sqrt{2}} \left( 1 -1 \right) \left( 0 -i \right) \frac{1}{\sqrt{2}} \left( 1 -1 \right) = \frac{1}{2} (1 \cdot i + (-1) \cdot i) = 0, \\
\langle Z \rangle &= \frac{1}{2} (\langle 0|0\rangle \langle 0|0\rangle - \langle 1|1\rangle \langle 1|1\rangle) = 0.
\end{align*}

These expectation values obviously depend on the state. The measurement with X is deterministic (the outcome is always -1, otherwise the expectation value would be greater than -1) whereas measuring with Y or Z is probabilistic. Though the state of the system is well-known, quantum mechanics can yield an intrinsic probabilistic property.

A system of multiple quantum components forms a new quantum system. If we have two components in states \( |\psi\rangle \) and \( |\psi'\rangle \), the entire system is described by the tensor product of the individual states \( |\psi\rangle \otimes |\psi'\rangle \). This is often shortened to \( |\psi\rangle |\psi'\rangle \) or \( |\psi\psi'\rangle \). The new state lives in the tensor product of the individual Hilbert spaces. Consider for example the two qubits \( |\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \) and \( |\psi'\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \). Together they form a system \( |\psi\rangle \otimes |\psi'\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle) \).

There are also systems of two qubits that cannot be decomposed into two smaller systems. For instance, the singlet state \( \frac{|01\rangle - |10\rangle}{\sqrt{2}} \) cannot be decomposed.

**Proof.** A product state of two qubits is a tensor product of two states \( \alpha |0\rangle + \beta |1\rangle \) and \( \gamma |0\rangle + \delta |1\rangle \). The tensor product is \( \alpha \gamma |00\rangle + \alpha \delta |01\rangle + \beta \gamma |10\rangle + \beta \delta |11\rangle \). This should be equal to \( \frac{|01\rangle - |10\rangle}{\sqrt{2}} \). Thus, \( \alpha \gamma = \beta \delta = 0 \). But then, \( \alpha \delta = 0 \) or \( \beta \gamma = 0 \) and the composite state cannot be equal to the singlet state. \( \square \)

Two qubits that form a state which is not a product state are called *entangled*. To describe the state of one of the qubits in an entangled state, we need to introduce the density operator.

**Definition 1.0.2.** A quantum system that is in each of the states \( |\psi_i\rangle \) with probability \( p_i \) can be described by the density operator

\[ \rho \equiv \sum_i p_i |\psi_i\rangle \langle \psi_i |. \]

The density operator of the singlet state is \( \rho = (\frac{|01\rangle - |10\rangle}{\sqrt{2}})(\frac{|01\rangle - |10\rangle}{\sqrt{2}}) \). In this case, \( \rho \) is a *pure state*. We can distinguish *pure* states from *mixed* states by computing \( \text{tr}(\rho^2) \equiv \sum_i \langle i | \rho^2 | i \rangle \) for any orthonormal basis \( \{|i\rangle\} \) which is the trace of the operator \( \rho^2 \).

**Lemma 1.0.3.** For every density operator \( \rho \), \( \text{tr}(\rho^2) \leq 1 \) with equality if and only if \( \rho \) is a pure state.

**Proof.** The density operator can be expressed in its orthonormal eigenbasis \( \{|\phi_i\rangle\} \) with eigenvalues \( \lambda_i \) by \( \rho = \sum_i \lambda_i |\phi_i\rangle \langle \phi_i |. \) Then, \( \text{tr}(\rho^2) = \sum_i \lambda_i^2 \). Because \( \rho = \rho^\dagger \) and \( \text{tr}(\rho) = 1 \), it follows \( 0 \leq \lambda_i \) and \( \text{tr}(\rho^2) \leq 1 \) with equality if and only if there is a \( j \) with \( \lambda_j = \delta_{ij} \), i.e. \( \rho \) is a pure state. \( \square \)
CHAPTER 1. INTRODUCTION

In case of the singlet state, we have \( \rho = \frac{1}{2}(|01\rangle - |10\rangle)(\langle 01| - \langle 10|) \). Let \( A, B \) be the particles in this state. Each subsystem can be described by a partial trace \( \rho_A = \text{tr}_B(\rho), \rho_B = \text{tr}_A(\rho) \) which is defined by \( \text{tr}_B(a_1 \langle a_2| \otimes |b_1\rangle \langle b_2|) \equiv a_1 \langle a_2| \text{tr}(|b_1\rangle \langle b_2|) \) where \( |a_1\rangle, |a_2\rangle \) are vectors of the subsystem \( A \) and \( |b_1\rangle, |b_2\rangle \) are vectors of the subsystem \( B \). In case of the singlet state, 

\[
\rho_A = \frac{1}{2}(|1\rangle \langle 1| + |0\rangle \langle 0|), \rho_B = \frac{1}{2}.
\]

In both subsystems, we obtain \( \text{tr}(\rho^2) = \frac{1}{2} \). That means, each particle on its own is in a mixed state, although the entire system is in a pure state. We do not know the subsystems’ states exactly. That is the characteristic of entanglement. On the contrary, a product state \( |\psi\rangle \otimes |\sigma\rangle \) can be described by the density operator \( \rho = \rho_A \otimes \rho_B = |\psi\rangle \langle \psi| \otimes |\sigma\rangle \langle \sigma| \).

Measuring a subsystem described by a density operator \( \rho_A \) can be described as for pure states. The measurement yields outcome \( m \) with probability \( p(m) = \text{tr}(M_m^\dagger M_m \rho_A) \) and the state after the measurement is \( \rho'_A = \frac{M_m \rho_A M_m^\dagger}{\text{tr}(M_m^\dagger M_m \rho_A)} \). This corresponds to a measurement by \( \{M_m \otimes 1\} \) on the entire system \( \rho \) with the same probabilities \( p(m) \) and the state \( \rho' = \frac{(M_m \otimes 1) \rho (M_m \otimes 1)^\dagger}{p(m)} \) after the measurement.

**Example 1.0.4.** Consider a Pauli measurement \( Z = |0\rangle \langle 0| - |1\rangle \langle 1| \) on the first qubit of the singlet state \( \rho = \frac{1}{2}(|01\rangle - |10\rangle)(\langle 01| - \langle 10|) \). The measurement yields \( \pm 1 \) each with probability \( \frac{1}{2} \) and the state after the measurement corresponds to the outcome:

\[
\rho'_0 = |01\rangle \langle 01|, \rho'_1 = |10\rangle \langle 10|
\]

Entanglement is one of quantum physics’ most astonishing properties, which cannot be observed in classical systems, that is still not fully understood. One application, that yields surprising results, is superdense coding.

### 1.1 Superdense Coding

We assume that two parties, namely Alice and Bob, aim on transmitting some classical information. Suppose Alice possesses two classical bits that she wants to communicate to Bob. Beforehand they have created the singlet state \( |\psi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \), each of them owning one of the qubits.

![Figure 1.1: Superdense coding between Alice and Bob.](image)
1.2. BELL INEQUALITY

Alice manipulates her qubit by applying one of the four Pauli matrices $\sigma_0 = 1$, $\sigma_1 = X$, $\sigma_2 = Y$, and $\sigma_3 = Z$, i.e. applying $\sigma_k \otimes 1$ on the entire system. Her choice depends on her classical bits ($ij \rightarrow \sigma_{2i+j}$). Afterwards, the state of the entire system is

$$
00 \rightarrow |\psi_{00}\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle),
$$

$$
01 \rightarrow |\psi_{01}\rangle = \frac{1}{\sqrt{2}}(|11\rangle - |00\rangle),
$$

$$
10 \rightarrow |\psi_{10}\rangle = \frac{i}{\sqrt{2}}(|11\rangle + |00\rangle),
$$

$$
11 \rightarrow |\psi_{11}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle),
$$

because Bob does not manipulate his qubit. These states form an orthonormal basis. Thus, when Alice sends Bob her qubit, he can perform a measurement in this basis and distinguish these states. The measurement outcome then directly corresponds to the two classical bits that Alice wanted to transmit. As we have seen, it is possible to transmit two classical bits by only transmitting one qubit using entanglement.

1.2 Bell Inequality

Superdense coding shows that quantum mechanics can be used to get unexpected results. But what exactly makes quantum systems exceptional and how can these be distinguished from classical systems? Bell inequalities, named after the inventor of the first ideas on how to distinguish classical and quantum physics, give one answer to this question.

To derive one of these inequalities, we assume that a classical system is characterized by physical properties $Q, R, S, T$. That is, these properties exist without observation. The measurement of these features merely reveals them. This reasoning follows from common sense since classically, looking at a ball does not determine its position but merely reveals it to the observer. We consider Alice and Bob examining such a system consisting of two particles that are prepared by a third party. Both Alice and Bob decide whether they measure $Q$ or $R$, $S$ or $T$ respectively by flipping a coin after each of them received his/her particle. They should do their measurements simultaneously. That means, they are far enough away from each other such that they are separated by a space-like interval. Special relativity tells us that this prevents a causal relationship between the two measurements.

For simplicity we assume that each of the variables $Q, R, S, T$ is either $\pm 1$. Let $p(q,r,s,t)$ be the probability that the prepared particles have properties $Q = q, R = r, S = s, T = t$ and consider the term

$$QS + RS + RT - QT = (Q + R)S + (R - Q)T.$$
CHAPTER 1. INTRODUCTION

It is easy to see that this quantity is either $\pm 2$. Thus, the expectation value, denoted in Dirac-notation $\langle \cdot \rangle$, of this quantity obeys

$$\langle QS + RS + RT - QT \rangle = \sum_{q,r,s,t} p(q,r,s,t) \cdot (qs + rs + rt - qt) \leq \sum_{q,r,s,t} p(q,r,s,t) \cdot 2 = 2$$

Analogously, we get $\langle QS + RS + RT - QT \rangle \geq -2$ and by linearity of the expectation value, we have derived the Bell-CHSH inequality which is named after its inventors John Clauser, Michael Horne, Abner Shimony, and Richard Holt:

**Theorem 1** (CHSH-Inequality [7]). A classical system with properties $Q, R, S, T$, all $\pm 1$ with some probability, satisfies the inequality

$$|\langle QS \rangle + \langle RS \rangle + \langle RT \rangle - \langle QT \rangle| \leq 2.$$

In contrary to a classical system, we now assume that the two particles are in the singlet state $|\psi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$, Alice owning the first and Bob the second one.

The indices indicate which particle is measured. Now, the values for $Q$, $R$, $S$, and $T$ are the eigenvalues of the measured operators. Alice can choose $Z$ or $X$, Bob can choose $-Z - X\sqrt{2}$ or $Z - X\sqrt{2}$. Measuring $Q = Z$ and $S = -\frac{Z - X}{\sqrt{2}}$ yields the following expectation value
\[ \langle QS \rangle = \langle Z_1 \otimes -\frac{Z_2 + X_2}{\sqrt{2}} \rangle \]
\[ = -\frac{1}{\sqrt{2}} \left( \langle Z_1 \otimes Z_2 \rangle + \langle Z_1 \otimes X_2 \rangle \right) \]
\[ = -\frac{1}{\sqrt{2}} \left( \frac{\langle 01 \rangle - \langle 10 \rangle}{\sqrt{2}} \left( \frac{-\langle 01 \rangle + \langle 10 \rangle}{\sqrt{2}} \right) + \frac{\langle 01 \rangle - \langle 10 \rangle}{\sqrt{2}} \left( \frac{\langle 00 \rangle + \langle 11 \rangle}{\sqrt{2}} \right) \right) \]
\[ = -\frac{1}{\sqrt{2}} \]

Analogously, it follows that \( \langle RS \rangle = \langle RT \rangle = \frac{1}{\sqrt{2}} \) and \( \langle QT \rangle = -\frac{1}{\sqrt{2}} \). Thus, \( |\langle QS \rangle + \langle RS \rangle + \langle RT \rangle - \langle QT \rangle| = 2\sqrt{2} \); the CHSH-inequality is violated. Therefore, it can be used to distinguish between systems that can be simulated classically and quantum systems that do not allow such a simulation. This result has been examined by experimenters and it has been shown that Nature does not obey the CHSH-inequality. Hence, at least one of the following two assumptions that were used in our common sense reasoning is not correct:

- The assumption that the physical properties are characteristics of the system and merely revealed by an observer. This is called *realism*.
- The assumption that there is no causal relationship between the two measurements because Alice and Bob perform their measurements simultaneously, i.e. with a space-like distance. This is called *locality*.

As a conclusion, we keep the locality and drop realism. Measuring some property of a quantum state does change the state. Our formulation of quantum mechanics is compatible with locality. There is no causal relationship between space-like separated particles.

Let \( Q, R, S, T \) be any observable with eigenvalues \( \pm 1 \). Thus, \( Q^2 = R^2 = S^2 = T^2 = 1 \). It follows that

\[ (Q \otimes S + R \otimes S + R \otimes T - Q \otimes T)^2 = (Q \otimes (S - T) + R \otimes (S + T))^2 \]
\[ = 1 \otimes ((S - T)^2 + (S + T)^2) + (QR - RQ) \otimes (ST - TS) \]
\[ = 4 \cdot 1 + [Q, R] \otimes [S, T] \]

where \([Q, R] \equiv QR - RQ\) is the commutator of the operators \( Q \) and \( R \). It is easy to see that \( \langle [Q, R] \rangle \leq \langle QR \rangle + \langle RQ \rangle \leq 2 \). Thus,

\[ \langle (Q \otimes S + R \otimes S + R \otimes T - Q \otimes T)^2 \rangle \leq 4 + 2 \cdot 2 = 8. \]

Taking the square root on both sides yields

**Theorem 2** (Tsirelson’s Bound [6]). *The maximum possible violation of the CHSH-inequality in quantum mechanics is \( |\langle QS \rangle + \langle RS \rangle + \langle RT \rangle - \langle QT \rangle| \leq 2\sqrt{2} \). This bound is tight as we have seen in our setup using the singlet state.*

The question has arisen whether there is a (theoretical) device violating Tsirelson’s bound.
Chapter 2

Non-local Boxes

In this chapter, the PR-box is introduced as a device that exceeds Tsirelson’s bound. It is named after its inventors Sandu Popescu and Daniel Rohrlich [17].

Definition 2.0.1. A non-local box is a device with two inputs and two outputs each of them providing one bit. One input ($x$) and one output ($a$) is held by Alice, the other input ($y$) and output ($b$) by Bob. The probability $P(ab|xy)$ determines the output behavior of the box. Both Alice and Bob should be able to extract their output from the moment they provide their input.

![Diagram of non-local box](figure21.png)

Figure 2.1: Setup of a non-local box.

From now on, we use bits to specify inputs and outputs instead of ±1 values. We can apply the CHSH-inequality to this setup by mapping the output bit 0 to the outcome +1 and 1 to −1. The input bits provide, according to the previous chapter, the measurement choices of Alice and Bob. In other words, Alice decides with her input bit $x$ whether she wants to measure property $Q$ or $R$, Bob similarly decides with his input bit $y$ whether he wants to obtain information about $S$ or $T$. The expectation value can therefore be computed by $X_{xy} = P(00|xy) + P(11|xy) − P(01|xy) − P(10|xy)$ for $x, y \in \{0, 1\}$. In this representation, the CHSH-value becomes

$$CHSH = \max_{x,y \in \{0,1\}} |X_{xy} + X_{xy} + X_{xy} - X_{xy}|$$

where $\bar{x}$ is the negation of $x$. Because the probabilities $P(00|xy), P(11|xy), P(01|xy), P(10|xy)$ sum up to 1 for each $x, y \in \{0, 1\}$, it follows that $|X_{xy}| \leq 1$ and the maximum possible CHSH-value is 4 which is more than Tsirelson’s bound.

In the classical system from the previous chapter, the properties $Q, R, S, T$ characterize the system and are merely revealed by observation. The measurement outcomes are only random
concerning the preparation process. Hence, \( P(ab|xy) = \sum_r P(r)(P_r(a|x) \cdot P_r(b|y)) \) where the bit string \( r \) encodes different preparations of the entire system each result of the preparation with probability \( P(r) \). The measurement outcome of Alice does only depend on her choice between measuring \( Q \) or \( R \) and similarly for Bob.

In the following, we repeat the results from the previous chapter using this formalism. With \( P_r(0|\alpha) + P_r(1|\alpha) = 1 \) for each \( \alpha \in \{x, \bar{x}, y, \bar{y}\}, r \), it follows that

\[
CHSH = \max_{x,y \in \{0,1\}} |X_{xy} + X_{x\bar{y}} + X_{\bar{x}y} - X_{\bar{x}\bar{y}}| \\
\leq \max_{x,y \in \{0,1\}, r} |P_r(0|x)P_r(0|y) + P_r(1|x)P_r(1|y) - P_r(0|x)P_r(1|y) - P_r(1|x)P_r(0|y) \\
+ P_r(0|x)P_r(0|\bar{y}) + P_r(1|x)P_r(1|\bar{y}) - P_r(0|x)P_r(1|\bar{y}) - P_r(1|x)P_r(0|\bar{y}) \\
+ P_r(0|\bar{x})P_r(0|y) + P_r(1|\bar{x})P_r(1|y) - P_r(0|\bar{x})P_r(1|y) - P_r(1|\bar{x})P_r(0|y) \\
- (P_r(0|\bar{x})P_r(0|\bar{y}) + P_r(1|\bar{x})P_r(1|\bar{y}) - P_r(0|\bar{x})P_r(1|\bar{y}) - P_r(1|\bar{x})P_r(0|\bar{y}))| \\
= 2 \cdot \max_{x,y \in \{0,1\}, r} |(|P(0|x) - 1)(P(0|y) + P(0|\bar{y}) - 1) + (2P(0|\bar{x}) - 1)(P(0|y) - P(0|\bar{y}))| \\
\leq 2 \cdot \max_{P(0|y), P(0|\bar{y}) \in [0,1]} ||P(0|y) + P(0|\bar{y}) - 1| + |P(0|y) - P(0|\bar{y})|| = 2
\]

as we would expect from our considerations of the first chapter. This bound is tight. It can be reached by \( P_0(0|x) = P_0(0|\bar{x}) = P_0(0|y) = P_0(0|\bar{y}) = 1 \) and \( P(r = 0) = 1 \).

The quantum setup using the singlet state \( |\psi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \) with measurement choices \( x \) for Alice and \( y \) for Bob with \( f : x \mapsto \begin{cases} Z_1 & \text{for } x = 0 \\ X_1 & \text{for } x = 1 \end{cases} \) and \( g : y \mapsto \begin{cases} -Z_2/X_2 & \text{for } y = 0 \\ 2Z_2/X_2 & \text{for } y = 1 \end{cases} \) is depicted in Figure 2.3.

Alice and Bob always measure an observable \( A \) with eigenvalues \( \pm 1 \) regardless of their measurement choices. \( A \) can be written as \( A = |\alpha\rangle\langle\alpha| - |\beta\rangle\langle\beta| \) for an orthonormal basis \( \{|\alpha\rangle, |\beta\rangle\} \).
Then,
\[
\frac{1 + (-1)^{\lambda} A}{2} = \frac{(|\alpha\rangle \langle \alpha| + |\beta\rangle \langle \beta|) + (-1)^{\lambda}(|\alpha\rangle \langle \alpha| - |\beta\rangle \langle \beta|)}{2} = \begin{cases} 
|\alpha\rangle \langle \alpha| = M_{+1} & \text{for } \lambda = 0 \\
|\beta\rangle \langle \beta| = M_{-1} & \text{for } \lambda = 1 
\end{cases}
\]
is a simple equation for the corresponding measurement operators $M_{(-1)^{\lambda}}$. Thus, the probability distribution for the quantum setup is given by

\[
P(ab|xy) = \langle \psi | \frac{1 + (-1)^{a} f(x)}{2} \otimes \frac{1 + (-1)^{b} g(y)}{2} |\psi\rangle.
\]

With $p_+ = \frac{2 + \sqrt{2}}{8}$ and $p_- = \frac{2 - \sqrt{2}}{8}$, the joint probability distribution is

\[
\begin{array}{c|cccc}
  & 00 & 01 & 10 & 11 \\
\hline
  11 & p_+ & p_- & p_+ & p_+ \\
  10 & p_- & p_+ & p_- & p_- \\
  01 & p_- & p_+ & p_- & p_- \\
  00 & p_+ & p_- & p_+ & p_+ \\
\end{array}
\]

Figure 2.4: Probability distribution of the quantum non-local box using the singlet state.

Therefore, $X_{00} = -X_{01} = X_{10} = X_{11} = \frac{1}{2} \sqrt{2}$ and $CHSH = 2 \sqrt{2}$ just as we would expect from the previous chapter.

In 1994, Sandu Popescu and Daniel Rohrlich introduced the PR-box as a device violating the CHSH-inequality maximally. It is defined by the following probability distribution.

**Definition 2.0.2 (Popescu-Rohrlich).** The PR-box is a non-local box

![Figure 2.5: Setup of a PR-box.](image)

with joint probability distribution $P(ab|xy) = \begin{cases} 
\frac{1}{2} & \text{for } a \oplus b = x \cdot y \\
0 & \text{else} 
\end{cases}$ where $\oplus$ is the addition modulo 2.

The PR-box maximizes the CHSH-value. It has $CHSH = 4$. 
CHAPTER 2. NON-LOCAL BOXES

2.1 Non-signaling Condition

The physical laws of motion due to special relativity prohibit information exchange faster than the speed of light. Because Alice and Bob should be able to extract their outputs instantaneously and the ends of a non-local box can be spatially separated arbitrarily far away from each other, the probability distribution \( P(ab|xy) \) must be chosen in such a way that communication is impossible. In other words, Alice should not be able to extract any knowledge about Bob’s input from her input and output and vice versa. This is captured in the following definition:

**Definition 2.1.1.** A non-local box is called non-signaling if

\[
P(a|x) \equiv \sum_{b \in \{0,1\}} P(ab|xy) = P(a|x) \quad \forall y \in \{0,1\}, \text{ and}
\]

\[
P(b|xy) \equiv \sum_{a \in \{0,1\}} P(ab|xy) = P(b|y) \quad \forall x \in \{0,1\}.
\]

This definition ensures that Alice cannot extract any information on \( y \) from her input \( x \) and output \( a \). Thus, the non-signaling condition restricts \( P(ab|xy) \) only in not allowing for instantaneous communication.

The classical system satisfies \( P(ab|xy) = \sum_r P(r)P_r(a|x)P_r(b|y) \) and hence, \( P(a|x) = \sum_r P(r)P_r(a|x)\) and \( P(b|y) = \sum_r P(r)P_r(b|y) \) are also satisfied. Thus, the classical setup is non-signaling.

The quantum non-local box uses a singlet state \( |\psi\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \). Alice and Bob can choose independently from each other in which orthonormal basis they want to measure their qubit. Let \( A = |\alpha\rangle \langle \alpha| - |\beta\rangle \langle \beta| \) be Alice’s measurement choice and \( B = |\gamma\rangle \langle \gamma| - |\delta\rangle \langle \delta| \) Bob’s. The probability for outcome \( ab \) is then given by \( P(ab) = \langle \psi| \frac{I_+(-1)^aA}{2} \otimes \frac{I_+(-1)^bB}{2} |\psi\rangle \). The marginal distribution is \( P(a) = \langle \psi| \frac{I_+(-1)^aA}{2} \otimes (\sum_{b \in \{0,1\}} \frac{I_+(-1)^bB}{2} ) |\psi\rangle = \langle \psi| \frac{I_+(-1)^aA}{2} \otimes 1 |\psi\rangle = P(a) \) and vice versa. According to the definition, the singlet state is a non-signaling device independently from Alice’s and Bob’s measurement choices.

It is straightforward to check that the PR-box is non-signaling as well:

\[
P(a|x) = \sum_{b \in \{0,1\}} \frac{1}{2} |(-1)^{a\oplus b} + (-1)^a| = \frac{1}{2} = P(a|x) \quad \text{and similarly} \quad P(b|xy) = \frac{1}{2} = P(b|y).
\]

All these devices are non-signaling and therefore allowed by special relativity. However, quantum mechanics does not allow the existence of PR-boxes because they exceed Tsirelson’s bound. Theorists study PR-boxes with the objective of understanding this restriction. Several surprising properties of these boxes have been discovered.

2.2 CHSH-Game

The CHSH-game is a game theoretic concept of introducing the PR-box. Alice and Bob both receive one bit \( a \) and \( b \) respectively and they are asked to output each one bit \( x, y \) such that
2.2. CHSH-GAME

\[ a \oplus b = x \cdot y \] with highest possible probability. Obviously, if they could communicate, this task would be trivial. That is why we only consider non-signaling devices, namely a classical system, a quantum non-local box and a PR-box.

![Diagram of the CHSH-game](image)

Figure 2.6: Setup of the CHSH-game.

A deterministic strategy for each of them cannot succeed with probability 1. Let \( f : 0 \rightarrow a_0, 1 \rightarrow a_1 \) be Alice’s deterministic strategy and \( g : 0 \rightarrow b_0, 1 \rightarrow b_1 \) Bob’s. A perfect strategy (succeeding with probability 1) would satisfy

\[
\begin{align*}
    a_0 \oplus b_0 &= 0, \\
    a_0 \oplus b_1 &= 0, \\
    a_1 \oplus b_0 &= 0, \\
    a_1 \oplus b_1 &= 1.
\end{align*}
\]

Summing up these equations (modulo 2), we obtain \( 0 = 1 \); a contradiction. We can at most satisfy three out of these four equations. Indeed, we are able to come up with a deterministic strategy for each subset containing three of these equations satisfying those. A classical strategy is a probabilistic mixture of deterministic strategies. Thus, we can probabilistically choose each of these deterministic strategies with equal probability and perform the task with probability \( \frac{3}{4} \) for each input which is therefore the maximum possible probability for a classical setup.

Suppose Alice and Bob share the state \( |\psi\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \). Let \( R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \) be a unitary rotation; Alice and Bob do manipulate their bit by applying \( R(-\frac{\pi}{16}) \) for \( x = 0, y = 0 \) and \( R(\frac{3\pi}{16}) \) for \( x = 1, y = 1 \) respectively. If Alice rotates over angle \( \theta_0 \), and Bob over angle \( \theta_1 \), they eventually share the state

\[
|\psi'\rangle = \frac{1}{\sqrt{2}}(\cos(\theta_0 + \theta_1)(|00\rangle - |11\rangle) + \sin(\theta_0 + \theta_1)(|01\rangle + |10\rangle)).
\]

By measuring in the orthonormal basis \{ \( \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \) \} corresponding to measurement outcome 0, 1 respectively, Alice and Bob succeed with probability \( \cos^2(\frac{\pi}{8}) > \frac{3}{4} \) when providing their measurement outcomes as outputs. This is the best, Alice and Bob can achieve using a quantum non-local box.
Proof. Suppose Alice and Bob, using a quantum system, succeed for each input with a probability $P(a \oplus b = xy) > \cos^2\left(\frac{\pi}{8}\right)$. Then,

$$X_{00}, X_{01}, X_{10} > 2\cos^2\left(\frac{\pi}{8}\right) - 1, \quad X_{11} < 1 - 2\cos^2\left(\frac{\pi}{8}\right)$$

With $\cos^2\left(\frac{\pi}{8}\right) = \frac{1}{4}(2 + \sqrt{2})$, we obtain $CHSH > 2\sqrt{2}$. This contradicts Tsirelson’s bound. Thus, Alice and Bob using a quantum non-local box can succeed with probability at most $\cos^2\left(\frac{\pi}{8}\right)$.

The PR-box can now be introduced as the unique device allowing Alice and Bob to perform this task perfectly without communication (and local bit flips). This result gives an insight into the power of PR-boxes. Several additional consequences of these non-local boxes are stated in the next chapter.
Chapter 3

Consequences of Superstrong Nonlocality

The introduction of PR-boxes has several surprising consequences. Some of the most remarkable applications are described in this chapter. In each of these applications, we consider two parties, Alice and Bob, having a common objective. We will examine how well they can perform the common task using classical systems, quantum systems and PR-boxes. To simplify matters, whenever Alice and Bob communicate, they send classical bits. They are not allowed to transfer qubits as we have seen in the superdense coding section.

This chapter discusses the exploitation of the properties of PR-boxes to reach trivial communication complexity for any distributed Boolean function, secure function evaluation and oblivious transfer.

The latter is used to introduce the concept of information causality. This helps in understanding why PR-boxes may never be realized (except in Minecraft). Nonetheless, research in this area can advance our understanding of quantum entanglement since non-local quantum systems can be considered as partial PR-boxes, as we have seen in the CHSH-game.

3.1 Communication Complexity

In high-performance computing, distributed processing is essential and even personal computers with four or more built-in processor units have become technical standard. The trade-off between high performance and communication overhead is a major challenge in highly distributed systems. In the following, we consider two parties computing a common Boolean function.

**Definition 3.1.1.** A distributed function \( f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\} \) is computed by two parties which we call Alice and Bob. Each of them holds a bit vector \( \vec{x}, \vec{y} \in \{0, 1\}^n \) and they need to communicate bits to compute \( f \) distributively if \( f \) depends non-trivially on \( \vec{x} \) and \( \vec{y} \). After following a certain protocol, Bob should determine the correct answer.

Depending on the available resources and the distributed function, a different amount of bits has to be communicated.

**Definition 3.1.2.** The communication complexity of a Boolean distributed function \( f \) is the minimum number of bits that have to be communicated between Alice and Bob to compute \( f \).
for an arbitrary input $\vec{x}, \vec{y}$. The usage of classical systems leads to the definition of classical communication complexity, usage of quantum systems to quantum communication complexity. The quantum system consists of an initial supply of entangled particles shared by Alice and Bob.

The inner product function

$$IP_n : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}, (x_1, \ldots, x_n, y_1, \ldots, y_n) \mapsto x_1 \cdot y_1 \oplus x_2 \cdot y_2 \oplus \cdots \oplus x_n \cdot y_n$$

is an example where the classical and the quantum communication complexity are lower-bounded by $\Omega(n)$ [8]. Obviously, in the case of $n$ communicated bits, Alice can simply transfer her input vector and Bob can compute the result locally. Therefore, communication complexity is an inherent non-trivial feature.

Having said this, the introduction of PR-boxes is sufficient to boil down necessary communication to only one bit. This is defined as trivial communication complexity since non-signaling devices cannot be used to transfer any bit of information. Therefore, a non-trivial distributed function will always have a communication complexity of at least 1 bit.

Wim van Dam showed in [20] that any distributed Boolean function has trivial communication complexity regarding the usage of sufficiently many PR-boxes.

**Theorem 3** (van Dam). The usage of $2^n$ PR-boxes suffices to reach trivial communication complexity for any distributed function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$.

**Proof.** Using the conjunctive normal form, each Boolean function can be constructed from the primitives $\text{AND} \land, \text{OR} \lor$ and $\text{NOT} \neg$. These primitives can be represented by multiplication, addition modulo 2 and the constant 1 which means 'true':

$$\begin{align*}
x \land y &\equiv x \cdot y, \\
x \lor y &\equiv x \oplus y \oplus x \cdot y, \\
\neg x &\equiv 1 \oplus x.
\end{align*}$$

Thus, each function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$ can be written as a finite sum modulo 2 of products of polynomials $P_i(\vec{x})$ and monomials $Q_i(\vec{y})$: $f(\vec{x}, \vec{y}) = \bigoplus_i P_i(\vec{x}) \cdot Q_i(\vec{y})$ by placing the monomials outside the brackets (compare example 3.1.3).

There are only $2^n$ monomials in $\vec{y} \in \{0,1\}^n$ because $y^n = y$ for each $y \in \{0,1\}$ and $n \in \mathbb{N}$, and in each monomial $y_i$ either appears or not. Hence, there are $2^n$ possible constructions for a monomial in $\vec{y} \in \{0,1\}^n$. Thus, each such function $f$ can be reduced to an inner product function with at the maximum $2^n$ summands $f(\vec{x}, \vec{y}) = \bigoplus_{i=1}^{2^n} P_i(\vec{x}) \cdot Q_i(\vec{y})$.

The probability distribution of a PR-box fulfills $P(a \oplus b = xy) = 1$. Thus, this inner product function with input vectors $P_i(\vec{x}), Q_i(\vec{y})$ for $i = 1, \ldots, 2^n$ can be computed by

$$f(\vec{x}, \vec{y}) = \bigoplus_{i=1}^{2^n} P_i(\vec{x}) \cdot Q_i(\vec{y}) \equiv \bigoplus_{i=1}^{2^n} a_i \oplus b_i \equiv (\bigoplus_{i=1}^{2^n} a_i) \oplus (\bigoplus_{i=1}^{2^n} b_i).$$

Alice’s side

Bob’s side
Alice computes the bit \( A = \bigoplus_{i=1}^{2^n} a_i \) on her side locally and sends it afterwards to Bob. Then, Bob determines \( A \oplus (\bigoplus_{i=1}^{2^n} b_i) \) locally on his side. This is \( f(\vec{x}, \vec{y}) \). Thus, using this protocol, Alice and Bob can compute any distributed function \( f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\} \) by using at the maximum \( 2^n \) PR-boxes and one bit of communication.

This protocol is used in the following example for the two-bit equality function which compares vectors \( \vec{x} \) and \( \vec{y} \) with \( \vec{x}, \vec{y} \in \{0, 1\}^2 \).

**Example 3.1.3.** The two-bit equality function can be expressed by \((x_1 \leftrightarrow y_1) \land (x_2 \leftrightarrow y_2)\) where \( x \leftrightarrow y \) is the equivalence operator which is true if and only if \( x \) and \( y \) are the same. We need to construct this function using multiplication, addition modulo 2 and the constant 1. Then, we can place the monomials in \( \vec{y} \in \{0, 1\}^2 \) outside the brackets:

\[
EQ(x_1, x_2, y_1, y_2) \equiv (x_1 \leftrightarrow y_1) \land (x_2 \leftrightarrow y_2) \\
\equiv (1 \oplus x_1 \oplus y_1) \cdot (1 \oplus x_2 \oplus y_2) \\
\equiv (1 \oplus x_1 \oplus x_2 \oplus x_1x_2) \cdot 1 \oplus 1 \cdot y_1y_2 \oplus (1 \oplus x_2) \cdot y_1 \oplus (1 \oplus x_1) \cdot y_2 \\
\equiv \bigoplus_{i=1}^{4} P_i(x_1, x_2) \cdot Q_i(y_1, y_2)
\]

Alice and Bob can now use \( P_i(x_1, x_2) \) and \( Q_i(y_1, y_2) \) as inputs for four shared PR-boxes and obtain vectors \( \vec{a} \in \{0, 1\}^2 \) and \( \vec{b} \in \{0, 1\}^2 \), respectively. These vectors satisfy \( EQ(x_1, x_2, y_1, y_2) \equiv (a_1 \oplus b_1) \oplus (a_2 \oplus b_2) \equiv (a_1 \oplus a_2) \oplus (b_1 \oplus b_2) \). Alice computes \( A = a_1 \oplus a_2 \) and sends this bit to Bob. Then, Bob computes \( A \oplus (b_1 \oplus b_2) \) which equals \( EQ(x_1, x_2, y_1, y_2) \).

In a classical setup, the communication complexity of the equality function is \( \Omega(n) \). Suppose, communicating \( n - 1 \) bits is sufficient. Then, Alice has to send the same message for two different bit strings \( \vec{x} \) and \( \vec{x}' \). Thus, Alice and Bob would evaluate \( EQ(\vec{x}, \vec{x}') = 1 \) in contrary to the definition.

### 3.2 Secure Function Evaluation

One may not always like to reveal one’s input vector \( \vec{x}/\vec{y} \) but nevertheless compute a distributed Boolean function \( f \). At best, Alice and Bob can only infer as much about each other’s input as they could from knowing \( f(\vec{x}, \vec{y}) \) and their own input. Any protocol can be compared to the following ideal scenario: Alice and Bob give away their input to a trusted third party. Once both inputs were received, the trusted third party computes \( f(\vec{x}, \vec{y}) \) and transfers this bit to both, Alice and Bob.

In the protocol that was proposed in the proof of theorem 3 only one bit of information is transferred between the two parties. This can be either 0 or 1 and hence, Bob can only infer whether \( f(\vec{x}, \vec{y}) = 0 \) or \( f(\vec{x}, \vec{y}) = 1 \) when both of them stick to the protocol.

Consider for example an ATM where you have to input your pin. The pin must be compared to the corresponding value in the database of the ATM. The function that needs to be evaluated...
is the equality function. Unfortunately, this ATM is not surely trustworthy and you don’t want to give away your pin. Using trustworthy, i.e. real, PR-boxes, you only have to give away one bit of information according to the protocol mentioned above. In this setting, you are Alice and the ATM represents Bob. Certainly, this communicated bit can only hold the information of whether your pin was right or wrong. Thus, the pin is kept save while at the same time a trustworthy ATM has the chance to verify the pin. It can be checked whether or not the pin was right without revealing more information than necessary.

3.3 Oblivious Transfer

As seen in the previous section, PR-boxes can be used to construct cryptographic primitives. Oblivious transfer is another example for this usage. Oblivious transfer is a fundamental problem in cryptography. For instance, it can be used as a primitive to construct secure multiparty computation.

**Definition 3.3.1.** *One-out-of-two oblivious transfer* is a device with two inputs on Alice’s side and one input and one output on Bob’s side. Alice provides two bits $x_0$ and $x_1$ and Bob decides which of them he would like to know ($x_c$). Bob must not learn anything about $x_{1-c}$ and Alice must not learn $c$ in the process.

![Figure 3.1: One-out-of-two oblivious transfer.](image)

Wolf and Wullschleger proposed the following protocol in [22] to get one-out-of-two oblivious transfer from PR-boxes. Alice inputs $x = x_0 \oplus x_1$ and Bob $c$ into a shared PR-box.

![Figure showing the protocol](image)

Alice receives output $a$ and sends $m = x_0 \oplus a$ to Bob. Bob receives $m$ and $b$ and computes $y = m \oplus b = x_0 \oplus a \oplus b = x_0 \oplus [(x_0 \oplus x_1)c] = x_c$. In this setting, only one bit is communicated due to the non-signaling feature of the PR-box. Thus, Bob cannot learn more than $x_c$. On the other hand, Bob does not send any information to Alice. Thus, Alice cannot learn anything
This result may be an argument against the existence of PR-boxes. It seems strange that Alice can send one bit and Bob can choose which of the two bits he wants to extract from this bit. The bit carries seemingly two bits of information, although Bob cannot learn both of them.

### 3.4 Information Causality

Motivated by these results, M. Pawlowski et al. formulated the concept of information causality [15]. Let Alice and Bob share non-signaling devices. After setup, Alice receives \( n \) random bits \( x_0, x_1, \ldots, x_{n-1} \) and is allowed to send \( m \) bits to Bob. Bob on the other side receives a random value \( b \in \{0, 1, \ldots, n - 1\} \). The task for Bob is to output \( x_b \). It is measured how good Alice and Bob can perform in this setting. That means, for how many \( b \in \{0, 1, \ldots, n - 1\} \) Bob can output \( x_b \).

Information causality holds when Bob can learn at most \( m \) bits in this setting. For physical theories we may claim information causality. In fact, this postulate coincides with the Tsirelson bound [15]. Thus, research on PR-boxes may be more of a theoretic kind. Information causality could potentially be an information theoretic criteria that distinguishes physical theories from non-physical non-local theories. This would be a great step in understanding the principles of nature. The following example shows that information causality does not hold in a setting with PR-boxes.

**Example 3.4.1.** Consider a setting with \( n \) PR-boxes shared between Alice and Bob. The following protocol enables Alice and Bob to perform perfectly, i.e. Bob can output \( x_b \) independently from \( m \geq 1 \).

1. Bob encodes \( b \) in \( \vec{y} \in \{0, 1\}^n \) by choosing \( y_b = 1 \) and \( y_i = 0 \) for \( i \neq b \), so \( y_i = \delta_{ib} \) for all \( i = 0, 1, \ldots, n - 1 \).

2. Alice and Bob perform the inner product function \( IP(\vec{x}, \vec{y}) \) of \( \vec{x} \) and \( \vec{y} \). It is \( IP(\vec{x}, \vec{y}) = \oplus_{i=0}^{n} x_i \cdot \delta_{ib} = x_b \). This function can be computed with only one bit of communication by using \( n \) PR-boxes.
Chapter 4

Implementation in Minecraft

Minecraft is an open-world game mainly build out of cubes, so called blocks. These blocks are large scaled compared to the player and differ mostly in appearance. The players have no pre-defined aim but can change the world to fit their imagination by destroying and placing blocks. Different tools must be created to exploit the mineral resources. These, in turn, can be used to craft new objects. Beside from primitive cubes like grass, dirt, and wood, the game contains so called redstone circuits that can be used to simulate electric circuits and implement logical elements.

![A typical Minecraft scenery in third person perspective.](image)

Minecraft can be customized by programming or installing modifications. These modifications can include new blocks with different textures and behavior. Using this option, Minecraft even has reached the classroom to help pupils in getting familiar with scientific content [19]. One of these modifications is qCraft.
CHAPTER 4. IMPLEMENTATION IN MINECRAFT

4.1 QCraft

QCraft implements features of quantum mechanics into Minecraft. It was developed by Dan200 and some friends from Google, MinecraftEdu, E-Line Media and CalTech and published in October 2013. It was made open source with release 1.7.2 in July 2014. There is a MinecraftEdu version of the modification that is specifically designed for usage in classrooms. Special maps and a curriculum were designed to introduce quantum physics and computing concepts to students of grades 6-8. QCraft was designed to let the players experience quantum mechanics and its counter-intuitive consequences like entanglement. However, one must be aware that this modification is not in every aspect a realistic simulation of quantum mechanics.

The fundamental block coming with qCraft is the quantum ore which is generated like any other ore throughout the Minecraft world. When mined, it drops quantum dust similar to redstone dust. This can be used to give quantum properties to ordinary blocks. Concepts included are observer dependent block, quantum block, entangled block, and quantum computer. Every graphic in this section is taken from [https://sites.google.com/a/elinemedia.com/qcraft/wiki/qcraft/blocks-and-items](https://sites.google.com/a/elinemedia.com/qcraft/wiki/qcraft/blocks-and-items).

4.1.1 Observer Dependent Block

An observer dependent block is crafted out of native blocks and the essence of observation, an item coming with qCraft, and takes character according to the viewing direction of a player. For instance, a player observing the block from the west determines its character according to the blocks that were used in the creation. The different characters come with different features like flammability, transparency and even existence. An observer dependent block can change its state only when there is no player that is looking at the block. Then, it is in a ‘superposition’ of the different states until it is observed again and changes its features according to the direction a player looks at it.

![Crafting recipe of the essence of observation](https://sites.google.com/a/elinemedia.com/qcraft/wiki/qcraft/blocks-and-items)

(a) (b)

Figure 4.2: The crafting recipe of the essence of observation out of four quantum dust (a) and an observer dependent block using the essence of observation (b).
This kind of block makes the player familiar with the concept of observer dependent measurements in quantum physics. A measurement in quantum mechanics affects the underlying system and therefore the possible outcomes. Consider for example the state \( |\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \). On the one hand, measuring in the basis \( \{ \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \} \) always yields the original state as the resulting state. On the other hand, measuring in the basis \( \{ |0\rangle, |1\rangle \} \) does yield either \(|0\rangle\) or \(|1\rangle\) each with probability \( \frac{1}{2} \). The direction, from which a player looks at an observer dependent block corresponds to the measurement choice, the different measurement outcomes correspond to the different characters the block can take.

In the qCraft setting, any measurement has a deterministic outcome. However, in quantum mechanics, there are two properties \( A \) and \( B \) that cannot be observed at the same time. Such properties \( A \) and \( B \) do not commute as operators, i.e. \([A, B] \neq 0\), and obey Heisenberg’s uncertainty principle. Certainly, we would say that a block cannot be sand and gold at the same time, so these features should also obey Heisenberg’s uncertainty principle. Otherwise, observing the block from different angles would not change its state. One can show that

\[ \text{Theorem 4 (Heisenberg). Two observables } A \text{ and } B \text{ satisfy } \Delta(A)\Delta(B) \geq \frac{|\langle\psi|[A,B]|\psi\rangle|^2}{2} \text{ for any quantum state } |\psi\rangle \text{ where } \Delta(\cdot) \text{ is the standard deviation.} \]

Observer dependent blocks do not satisfy Heisenberg’s uncertainty principle. They lack the intrinsic probabilistic feature of quantum mechanics. Fortunately, the qCraft modification also introduces a quantum block to address this problem.

### 4.1.2 Quantum Block

A quantum block is created similarly to an observer dependent block out of native blocks and the essence of superposition, which also comes with qCraft. The two native Minecraft blocks that are put in a line of sight in the recipe either appear in place of the quantum block with the same probability when the block is observed from one of the related directions. That means, a quantum block that is observed from the north will take the character of either the block that is placed south or north in the crafting recipe.
This kind of block makes the player familiar with the concept of indeterminism in quantum settings. This represents a quantum state, for example $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, and measurement choices, for instance $Z$ or $Y$. The different measurement outcomes are equally probable. However, in a quantum setup, after the measurement the resulting state is an eigenstate of the related observable. Thus, measuring the same observable afterwards yields the same measurement outcome with probability 1. This behavior is not implemented in qCraft. Quantum blocks do not change their states but only simulate the intrinsic probabilistic character of quantum mechanics.

### 4.1.3 Entangled Block

Two identical observer dependent blocks or quantum blocks together with the essence of entanglement can be used to craft two entangled blocks. These entangled blocks always take the same character, no matter where they were placed. They can only change this state when none of them is observed by any player.

![Crafting recipe](image)

Figure 4.4: The crafting recipe of the essence of entanglement out of four quantum dust and one essence of superposition (a), two entangled blocks using the essence of entanglement and two identical observer dependent blocks or quantum blocks (b), and adding another identical block to a group of entangled blocks, one of them representing the group in the recipe (c).

This kind of block is used to bring the concept of entanglement to the game. All of the entangled blocks take the same character as the one of them that is observed by a player. For two blocks, such a correlation can be expressed classically by the mixed state $\rho = \frac{1}{2}(|00\rangle \langle 00| + |11\rangle \langle 11|)$ and a measurement in the computational basis $\{|0\rangle, |1\rangle\}$. However, looking at the observed block from another direction should correspond to a measurement in another orthonormal basis $\{|\alpha\rangle, |\beta\rangle\}$ with $|0\rangle = a|\alpha\rangle + b|\beta\rangle$ and $|1\rangle = c|\alpha\rangle + d|\beta\rangle$. The state can be expressed in this basis by $\rho = \frac{1}{2}(|a|^2 + |c|^2)|\alpha\rangle \langle \alpha| + (|a|^2|b|^2 + |c|^2|d|^2)(|\alpha\beta\rangle \langle \alpha\beta| + |\beta\alpha\rangle \langle \beta\alpha|) + (|b|^2 + |d|^2)|\beta\beta\rangle \langle \beta\beta|)$. Obviously, there is no other than the computational basis that yields perfect correlation. Thus, classical correlation is not sufficient in describing the behavior of entangled blocks.

The perfect correlation of entangled blocks can also be compared to the quantum state $\rho = \frac{1}{2}(|00\rangle \langle 00| + |11\rangle \langle 11|)$ of two entangled qubits with measuring the observed qubit in the computational basis $\{|0\rangle, |1\rangle\}$. Hence, the state of the observed particle is given by $\rho_A = \frac{3}{2}$ which is a mixed state since $\text{tr}(\rho_A^2) = \frac{1}{2} < 1$. In another orthonormal basis $\{|\alpha\rangle, |\beta\rangle\}$, the state of the two qubits can be written as $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}((a^2 + c^2)|\alpha\alpha\rangle + (ab + cd)(|\alpha\beta\rangle + |\beta\alpha\rangle) + (b^2 + d^2)|\beta\beta\rangle)$. A pair of entangled blocks in this state measured in the basis $\{|\alpha\rangle, |\beta\rangle\}$ is not necessarily perfectly correlated, i.e. the individual qubits do not necessarily
show the same behavior. Moreover, the state of entangled blocks recovers whenever it is not observed. In contrast, the state of a real quantum system remains unchanged without any manipulation.

A singlet state $|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ can be used to get perfect anti-correlation in each possible basis. To see this, consider an orthonormal basis $\{|\alpha\rangle, |\beta\rangle\}$ with $|0\rangle = a|\alpha\rangle + b|\beta\rangle$ and $|1\rangle = c|\alpha\rangle + d|\beta\rangle$. Then, $|\psi\rangle = (ad - bc)\frac{1}{\sqrt{2}}(|\alpha\beta\rangle - |\beta\alpha\rangle)$ where $(ad - bc) = \det(U)$ for the change of basis matrix. Because $\{|0\rangle, |1\rangle\}$ and $\{|\alpha\rangle, |\beta\rangle\}$ are orthonormal bases, we know from linear algebra that $\det(U) = e^{i\theta}$ for some real $\theta$. For any measurement operator $M_m$, the outcome probabilities for a state $e^{i\theta}|\varphi\rangle$ are given by $\langle \varphi | e^{-i\theta} M_m^\dagger M_m e^{i\theta} | \varphi \rangle = \langle \varphi | M_m^\dagger M_m | \varphi \rangle$ and hence, $|\varphi\rangle$ and $e^{i\theta}|\varphi\rangle$ cannot be distinguished by any measurement. Thus, we can write $|\psi\rangle = \frac{1}{\sqrt{2}}(|\alpha\beta\rangle - |\beta\alpha\rangle)$ which shows perfect anti-correlation in any basis. As we have seen above, there is no state that yields perfect correlation, contrary to the qCraft implementation.

### 4.1.4 Quantum Computer

A quantum computer is crafted out of glass pane, iron ingot, and quantum dust. By constructing a matrix building out of obsidian, ice, glass blocks, and observer dependent blocks out of gold (in the corresponding cardinal direction) and obsidian (in each other direction), the player determines what area should be affected by the quantum computer. Then, the specified area is copied, so called quantized, and you can place it elsewhere by constructing an identical matrix at another point in the world and de-quantize the quantum computer there.

Two quantum computers can be entangled similarly to entangled blocks. Entangled quantum
computers can be used to teleport a specified area. This area is specified by a matrix as in the case of quantizing. The two related matrices must be of the exact same shape, so that the content can be swapped. This is done by energizing one of the entangled quantum computers.

Entangled quantum computers also can be used to construct quantum portals that use a different kind of matrix. Through these portals, a player can travel not only to different places within the world, but also between different servers and therefore worlds.

Though this implementation is far from realistic, it gives an introduction to the concept of quantum teleportation. Bennett et al. introduced quantum teleportation in [3]. Consider Alice and Bob sharing the singlet state $\ket{\psi} = \frac{1}{\sqrt{2}}(\ket{01} - \ket{10})$ with Alice owning particle 1 and Bob particle 2. Alice now wants to teleport the state $\ket{\psi_0} = a \ket{0} + b \ket{1}$ with $|a|^2 + |b|^2 = 1$ of a third particle. Alice can measure the state of her two particles using the orthonormal basis

$$\ket{\phi^\pm} = \frac{1}{\sqrt{2}}(\ket{01} \pm \ket{10})$$

$$\ket{\Phi^\pm} = \frac{1}{\sqrt{2}}(\ket{00} \pm \ket{11})$$

The complete state of all particles is given by

$$\ket{\Psi} = \ket{\psi_0} \otimes \ket{\psi} = \frac{a}{\sqrt{2}}(\ket{000} - \ket{010}) + \frac{b}{\sqrt{2}}(\ket{101} - \ket{110})$$

$$= \frac{1}{2}(\ket{\phi^-} \otimes (-a \ket{0} - b \ket{1}) + \ket{\phi^+} \otimes (-a \ket{0} + b \ket{1})$$

$$+ \ket{\Phi^-} \otimes (a \ket{1} + b \ket{0}) + \ket{\Phi^+} \otimes (a \ket{1} - b \ket{0}))$$.

Thus, Alice’s measurement projects Bob’s particle’s state on one of the following possibilities with equal probability.

$$- \ket{\psi_0} = \begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \ket{\psi_0}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \ket{\psi_0}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \ket{\psi_0}$$

After her measurement, Alice sends two classical bits to Bob to specify in which of these four states Bob’s particle has collapsed. Certainly, the outcome of her measurement yields this information as it yields one of the four orthonormal states $\ket{\phi^\pm}$ and $\ket{\Phi^\pm}$. Bob can now recover $\ket{\psi_0}$ by applying one of the above unitary transformations’ inverse on his particle.

As a remark, Alice does not own the state $\ket{\psi_0}$ anymore. Thus, the no-cloning principle, that forbids to clone a quantum state, is not violated.
4.1.5 Additional Features

In addition to the presented devices, quantum goggles, anti-observation goggles, and automated observers found their way into qCraft. Quantum goggles can be used to identify blocks of the qCraft modification that were placed in the world. This is helpful especially for observer dependent blocks which only change their characters when observed from another direction. Moreover, on a multiplayer server, an entangled block can only change its character when none of the players is observing any of the entangled blocks belonging to the related entanglement group. Observer dependent blocks and quantum blocks can also disappear as one of their possible states. In such a state, they only can be detected using quantum goggles. Anti-observation goggles prevent qCraft blocks from being observed by the player that wears them and therefore from changing their state. These can be used to construct certain puzzles where observation triggers a trap. Both quantum goggles and anti-observation goggles are equipped in the helmet slot.

Figure 4.8: The crafting recipe of quantum goggles using one quantum dust and two glass panes.

Figure 4.9: The crafting recipe of anti-observation goggles using one essence of observation and two glass panes (a) and automated observers using seven stones, one essence of observation, and one redstone dust (b).

The automated observer is a redstone device that can be used to trigger observations using redstone signals. It has to be placed exactly next to the block that should be observed. This is useful to cause events over large distances or automatically.

4.2 PR-box Modification

As described above, qCraft has many weaknesses in simulating quantum mechanics. There is considerable variety in manifestations of quantum mechanics like the structure of atoms and the binding of molecules. It is very difficult if not impossible to implement only a subset of these quantum effects into Minecraft accurately. There are infinitely many different bases one could choose to measure a qubit. Blocks in Minecraft are cubic; they only provide six different ways to look at them. There are infinitely many different quantum states to describe only one qubit,
not to mention a system of entangled qubits. However, one cannot introduce this variety into Minecraft since it is limited to big building blocks which on the one hand allows the players to design the world according to their wishes, but on the other hand prevents them from adding fine details.

It is very natural to bring the concept of PR-boxes into Minecraft. PR-boxes are discrete devices with two options for observation. A PR-box cannot be measured repeatedly and hence, there are no difficult changes in status as we would expect from a quantum state.

The PR-box modification brings the non-local boxes into Minecraft. The PR-boxes are designed similar to a diode, interacting with redstone signal. It is possible to construct electrical circuits from the redstone primitives: redstone dust, redstone torch, and levers.

A luminous line of redstone indicates a logical 1 a dark line a logical 0. It is possible to implement such primitives as NOT, AND, and OR. These primitives form a functionally complete set, i.e. any logical function can be constructed from these primitives.

One PR-box as described above is divided into two ordinary blocks. To create one of these blocks, a player needs two quantum computers. This choice was made to illustrate the super-quantum correlation of PR-boxes. The two blocks get correlated by using the essence of entanglement. Two correlated blocks form a PR-box as it was described throughout the thesis. In this setting, Alice and Bob each own one of the blocks and can place them in the Minecraft world independently.

The input can be specified using a redstone signal on the input side which is colored red. If there is no connected redstone circuit, the specified input will be 0. The output is given by
4.2. PR-BOX MODIFICATION

Figure 4.12: The crafting recipe of a PR-box block out of two quantum computers (a) and the entanglement creation using the essence of entanglement (b).

(a)  (b)

a redstone signal on the opposite side which is either turned on (1) or turned off (0). Alice receives her output by applying a right-click on her block, similarly for Bob. Hereafter, this process is referred to as energizing. After this ’measurement’, the energized block maintains its output for the rest of the game.

Figure 4.13: A PR-box before (a) and after (b) energizing using levers to power the inputs. When energized, the block’s color changes from blue to green.

(a)  (b)

Alice and Bob can energize their blocks independently. That means, if Alice energizes her block and Bob has not energized his block yet, the outcome \( a \) for Alice will be 0 or 1 each with probability \( \frac{1}{2} \). Bob’s output is then determined to be \( b \) satisfying \( a \oplus b \equiv x \cdot y \) for inputs \( x \) and \( y \). Note, that there is always an input specified. A PR-box which is not connected to redstone is equivalent to a block that is connected to a non-powered redstone conduit because these situations cannot be distinguished in Minecraft. By the same token, the output of a non-energized PR-block is always 0.

Each pair of blocks that form a PR-box are associated with a group. The information which blocks are correlated can only be examined in the inventory. After a PR-block is placed in the Minecraft world, the only chance to find out the correlation group is to re-collect the block and examine it in the inventory. This is not possible for energized

Figure 4.14: Examining the correlation group of a PR-box.
PR-blocks. These get destroyed when they get picked up. The non-local feature of a PR-box can only be exploited once. According to van Dam’s protocol for computing distributed functions, the number of used PR-boxes determines the number of super-quantum correlations.

The PR-boxes in Minecraft can be used completely analogously to their counterparts from theory. Figure 4.16 shows how the redstone primitives can be used for addition modulo 2. The depicted construction might be counter-intuitive but provides a space-saving design. The setup uses freely selectable ordinary Minecraft blocks. This is used to implement the inner product function \( x_1 \cdot y_1 \oplus x_2 \cdot y_2 \) which is necessary for van Dam’s protocol for distributive computing. The computation requires four PR-blocks and hence, two PR-boxes shared by Alice and Bob.

A similar structure can be used to compute the equality function

\[
EQ(x_1, x_2, y_1, y_2) \equiv (1 \oplus x_1 \oplus x_2 \oplus x_1 x_2) \cdot 1 \oplus 1 \cdot y_1 y_2 \oplus (1 \oplus x_2) \cdot y_1 \oplus (1 \oplus x_1) \cdot y_2 \\
\equiv \neg(x_1 \lor x_2) \cdot 1 \oplus 1 \cdot y_1 y_2 \oplus \neg x_2 \cdot y_1 \oplus \neg x_1 \cdot y_2.
\]

This function is useful for password queries. The equality function is evaluated securely. That means, Alice and Bob only learn whether or not their inputs are the same but nothing more.
4.2. PR-BOX MODIFICATION

Figure 4.17: Computing the equality function $EQ(x_1, x_2, y_1, y_2)$ distributively and securely using one communicated bit (top left: $y_1$, bottom left: $y_2$, top right: $x_2$, bottom right: $x_1$; output at the very bottom). Correlated PR-blocks are located opposite each other.

Oblivious transfer is implementable as well and is shown in figure 4.18. Alice only transfers one bit and Bob is able to retrieve any of the two bits $x_0$ and $x_1$ as he wishes but not both of them. The communicated bit is transferred through the redstone conduit at the very bottom. The redstone signal weakens over distance and goes off after a distance of fifteen blocks. Fortunately, this is enough for this construction. The signal can be reinforced with a double NOT gate or a repeater which is a native Minecraft building block. Alice provides her input $x_0 \oplus x_1$ at the left and computes $a \oplus x_0$ where $a$ is her output at the bottom left. Then, she sends this bit to Bob who can calculate $x_c \equiv a \oplus b$ at the right where $b$ is his output and $c$ is his input. Bob can energize his PR-block whenever he likes to do so, even after receiving $a \oplus x_0$. This bit carries the information of potentially two bits. This surprising behavior led to the concept of information causality.

Figure 4.18: Oblivious transfer (at the very top from left to right: $x_1$ and $x_0$, in the middle: $c$ and at the very right $x_c$).

4.2.1 Spatial Partition

The PR-box was introduced as a black box which is intrinsically partitioned in two parts owned by two agents, namely Alice and Bob. To address this requirement of spatial independence, the PR-box is implemented as two ordinary Minecraft blocks. These get correlated analogously to
the entangled blocks by using the essence of entanglement and two uncorrelated boxes. This correlation process requires both parts at the same place, exactly as it is needed when creating a singlet state. Afterwards, both blocks can be spatially divided without any restrictions. Thus, two players can exploit all the described consequences of non-local boxes in Minecraft using the PR-box modification.

4.2.2 Re-use

Minecraft, after all, is still a game, so a modification should bring playable elements to the game. It should make sense to use new blocks in Minecraft because they simplify certain actions. The behavior of a PR-box is mostly fixed by its probability distribution, but it is reasonable to think about a re-use of the non-local boxes. From the point of view of a player, after usage the block should not be useless to some extent, since it must be recreated to process another computation. However, re-using non-local boxes brings up a problem of timing issues. That you can only use it once, seems to be an intrinsic feature of the PR-box, but I have made some attempts to bypass this problem.

In an early version of the PR-box modification both parts of a non-local box were updated according to the related probability distribution whenever one of them was energized. That means, the correlated blocks satisfy \( a \oplus b \equiv x \cdot y \) at any time. Whenever one of the inputs is altered, the outputs change accordingly. The two possible outcomes are equally distributed. Thus, if Alice modifies her input, Bob’s output could change and vice versa as depicted in figure 4.19.

Unfortunately, in this case the non-signaling property does no longer hold. Alice and Bob can utilize this setting to communicate. Consider Alice and Bob owning one part of such a non-local box each. Suppose Alice wants to send a bit to Bob. If Alice energizes her part of the PR-box, the output of Bob’s part does change with probability \( \frac{1}{2} \). To understand that this setup can be used for communication, we need the following definitions.
4.2. PR-BOX MODIFICATION

**Definition 4.2.1.** The *mutual information* of two discrete random variables $X$ and $Y$ is

$$I(X, Y) = \sum_{x \in X} \sum_{y \in Y} p_{X,Y}(x, y) \log_2 \frac{p_{X,Y}(x, y)}{p_X(x)p_Y(y)},$$

where $p_X(x) = \sum_{y \in Y} p_{X,Y}(x, y)$ and vice versa.

The *channel capacity* is defined by $C = \max_{p_X} I(X, Y)$, where $p_X$ describes the input distribution and $p_Y$ the output distribution.

Alice and Bob agree on a time interval beforehand. Then, Alice encodes her bit in whether she changes her input in this certain time interval or she does not. If her bit is 0, she will maintain her input, otherwise she will change it. Bob waits the time interval. If his output does change, he will know that Alice wanted to communicate the bit 1. If it does not, he cannot be sure. The depicted setup corresponds to the noisy communication channel

![Diagram](image)

Figure 4.20: Noisy channel using repeated energizing.

Alice’s coding uses the bit 0 with probability $p_X(0) = p$ and the bit 1 with probability $p_X(1) = 1 - p$ when encoding a random bit string. The joint probability distribution is

$$p_{X,Y} : \{0, 1\}^2 \to [0, 1], (0, 0) \mapsto p, (0, 1) \mapsto 0, (1, 0) \mapsto \frac{1-p}{2}, (1, 1) \mapsto \frac{1-p}{2}.$$ 

Thus, the output distribution is $p_Y(y) = \sum_x p_{X,Y}(x, y) : \{0, 1\} \to [0, 1], 0 \mapsto \frac{1+p}{2}, 1 \mapsto \frac{1-p}{2}$.

With this model, it is possible to compute the channel capacity:

$$C = \max_{p \in [0,1]} [p \log_2 \frac{2}{1+p} + \frac{1-p}{2} \log_2 \frac{1}{1+p} + \frac{1-p}{2} \log_2 \frac{1}{1-p}]$$

with $C = \log_2 \frac{5}{4}$ at $p = \frac{3}{5}$. This channel capacity is directly related to the communication rate achievable with this channel [9].

**Theorem 5** (Channel coding theorem). All communication rates $R$ with $R < C$ are achievable, i.e. there exists a code sequence for each block length $n$ such that the maximum probability of error $\lambda \to 0$. 

CHAPTER 4. IMPLEMENTATION IN MINECRAFT

Thus, our channel does allow communication since \( C > 0 \). Alice can find a coding scheme that allows her to transfer \( \log_2 \frac{5}{4} \approx 0.32 \) bits per time interval. This setup violates the intrinsic non-signaling feature of non-local boxes. Therefore, this implementation would not accurately represent non-local boxes.

The final implementation of the PR-box modification forbids re-use of PR-boxes. After one part of the box is energized, it becomes an ordinary block whose output is fixed obeying the non-local correlation of the inputs and outputs at the time of energizing. This implementation most accurately illustrates PR-boxes as an analogue to singlet states with super-quantum correlation. They can only be measured once and afterwards maintain their measurement outcome. As described above, Alice and Bob can energize their boxes to provide their inputs. When one of them does not provide any input, or even does not place his/her PR-block, the other one receives a random output. This is necessary to cover the possibility that one eventually does place his part of the PR-box and demands an output. This output is then determined by the PR-box correlation \( a \oplus b \equiv x \cdot y \).

4.2.3 Timed Synchronization

Exploring the consequences of superstrong nonlocality is quite laborious when one has to recreate and replace the PR-boxes for each computation. To automate this process of recreation and replacing, I introduce a timed version of the PR-box. Its energizing process is independent from the players. That means, the non-local boxes update both at specific points in time (every five seconds) autonomously. A timed PR-block can be crafted out of two ordinary PR-blocks and a clock to illustrate the time dependency. Two timed PR-blocks can be correlated by using the essence of entanglement, analogously to the correlation of two ordinary PR-blocks.

![Figure 4.21: The crafting recipe of a timed PR-box using a clock and two ordinary PR-boxes (a) and its appearance in the Minecraft world (b).](image)

Certainly, communication is not possible through this protocol since the output of each part of one PR-box changes at these specific points with probability \( \frac{1}{2} \) independently from the inputs. At the same time, the two parts always obey the right correlation after an update step. If the non-local box is updated regularly, the players will know exactly when the correlation is fulfilled. Redstone circuits can be adjusted dynamically and the results can be observed immediately. Another improvement is the opportunity to use non-local boxes for computations that
4.2. **PR-BOX MODIFICATION**

are controlled by a timer. The drawback of this method is that the timer and the frequency of updates has to be coupled. Perhaps this could be improved by a manipulation of the frequency via additional redstone signals. Unfortunately, this method is likely to violate the non-signaling feature. A frequency manipulation is possible by using the cheat "/prtimer [ticks]". 20 ticks are typically equivalent to one second. Although this implementation does not illustrate PR-boxes as accurately as the usual PR-box implementation, it is very helpful in experimenting with the super-quantum correlation.

### 4.2.4 Technical Implementation

Because of qCraft being made open-source, I could build the PR-modification on top of it. It uses the same correlation infrastructure. The elements in Minecraft follow a hierarchic structure and the implementation of the PR-box modification required adjustments at each level.

The top layer is a block-class specifying the general behavior of the block instances placed in the world. The appearance is specified, the mining requirements and the behavior under different actions like placing, mining, and destroying is determined. Additionally to the qCraft blocks, I had to add a redstone functionality. Using the native Minecraft code for a diode, I adjusted the behavior and uncoupled the in- and output to satisfy the super-quantum correlation.

The bottom layer consists of a tileEntity-class and an item-class. For every instance of a block in the world, there is a tileEntity. The tileEntity contains metadata like the facing direction and the color of different wool types. In the PR-box modification, it includes the correlation relation that must be saved and changed whenever the block instance changes, for example when the block is destroyed, or placed, or the redstone signal changes. The item-class comes into effect, when a destroyed tileEntity instance drops a representative of the related block. This representative is an item. It can be collected by the player and stored in his/her inventory. In qCraft and the PR-box modification, an item carries the correlation information. In the inventory, the correlation group of a PR-box item can be examined.

The correlation itself is managed by the main class of the modification which therefore constitutes a layer above the block-class layer. It is responsible for saving in-game information to a file and recovering the correlation at a restart. It also controls the timer for the timed version of the PR-box.
Chapter 5

Summary

In 1994, Sandu Popescu and Daniel Rohrlich developed the concept of non-local boxes that obey the non-signaling condition yet yielding a maximal CHSH-value \([17]\). In this work, the theory of PR-boxes is investigated and they were implemented in the sandbox video game Minecraft.

First, an introduction to quantum computing is given and the notion of Bell inequalities is described. These inequalities are applied to an information theoretic setting and the non-signaling condition is examined. This motivates the definition of PR-boxes in two different ways: maximizing the CHSH-value and perfectly performing the CHSH-game.

Distributed computation and cryptographical challenges become easier with a sufficient number of PR-boxes. It was shown by van Dam that communication complexity becomes trivial \([20]\). Secure function evaluation and oblivious transfer are easily implementable with non-local boxes. Any of these consequences of superstrong nonlocality and probably any application that is discovered in the future can be simulated in the PR-box modification that brings non-local boxes into Minecraft.

The qCraft modification suffers from inherent inaccuracies because quantum mechanics is not transferable one to one to the discrete Minecraft world lacking fine details. An accurate implementation of PR-boxes matches the nature of the big-block world. Solely playability is an issue due to the forbidden re-use. The timed version of the PR-box allows fast trial and error in building logical circuits that use non-local boxes. Certainly, the timed PR-box is not that precise as an analogue to a singlet state with super-quantum correlation; but it enables a fast and intuitively accessible way of interacting with the concept of PR-boxes.
Bibliography


Appendix A

PR-box Modification Code

This appendix includes the Java-code that makes up the PR-box modification. The code is partially taken from the qCraft modification, especially the correlation infrastructure.

Main Class

```java
package tys.prbox;

import static net.minecraftforge.oredict.RecipeSorter.Category.SHAPED;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
import cpw.mods.fml.common.FMLCommonHandler;
import cpw.mods.fml.common.Mod;
import cpw.mods.fml.common.Mod.EventHandler;
import cpw.mods.fml.common.Mod.Metadata;
import cpw.mods.fml.common.ModMetadata;
import cpw.mods.fml.common.event.FMLInitializationEvent;
import cpw.mods.fml.common.event.FMLPreInitializationEvent;
import cpw.mods.fml.common.event.FMLServerStartingEvent;
import cpw.mods.fml.common.eventhandler.SubscribeEvent;
import cpw.mods.fml.common.gameevent.TickEvent.ServerTickEvent;
import cpw.mods.fml.common.registry.GameRegistry;
import dan200.QCraft;
import dan200.qcraft.shared.ConnectionHandler;
import net.minecraft.command.ServerCommandManager;
import net.minecraft.init.Items;
import net.minecraft.item.ItemStack;
import net.minecraft.nbt.CompressedStreamTools;
import net.minecraft.nbt.NBTTagCompound;
import net.minecraft.world.World;
import net.minecraftforge.common.MinecraftForge;
import net.minecraftforge.event.world.WorldEvent.Load;
import net.minecraftforge.event.world.WorldEvent.Save;
import net.minecraftforge.event.world.WorldEvent.Unload;
import net.minecraftforge.oredict.RecipeSorter;

@Module(modid = PRbox.MODID, version = PRbox.VERSION)
public class PRbox {
    public static final String MODID = "prbox";
    public static final String VERSION = "1.0";
    @Metadata
    public static ModMetadata meta;
    public static int timer = 100;
```
APPENDIX A. PR-BOX MODIFICATION CODE

```java
/* Blocks */
public static BlockPRbox prbox;
public static BlockPRbox prboxPower;
public static BlockPRboxActive prboxActive;
public static BlockPRboxActive prboxActivePower;
public static BlockTimedPRbox timedPRbox;
public static BlockTimedPRbox timedPRboxPower;

@EventHandler
public void preInit(FMLPreInitializationEvent event) {
    /* Register Blocks */
    prbox = (BlockPRbox) new BlockPRbox(false).setBlockName("PRbox").setBlockTextureName("prbox:prbox").setCreativeTab(QCraft.creativeTab);
    prboxPower = (BlockPRbox) new BlockPRbox(true).setBlockName("PRboxPower").setBlockTextureName("prbox:prbox");
    prboxActive = (BlockPRboxActive) new BlockPRboxActive(false).setBlockName("PRbox (active)").setBlockTextureName("prbox:prboxActive");
    prboxActivePower = (BlockPRboxActive) new BlockPRboxActive(true).setBlockName("PRboxPower (active)").setBlockTextureName("prbox:prboxActive");
    timedPRbox = (BlockTimedPRbox) new BlockTimedPRbox(false).setBlockName("TimedPRbox").setBlockTextureName("prbox:timedprbox");
    timedPRboxPower = (BlockTimedPRbox) new BlockTimedPRbox(true).setBlockName("TimedPRboxPower").setBlockTextureName("prbox:timedprbox");
    GameRegistry.registerBlock(prbox, ItemPRbox.class, "prbox");
    GameRegistry.registerBlock(prboxPower, ItemPRbox.class, "prboxPower");
    GameRegistry.registerBlock(prboxActive, "prboxActive");
    GameRegistry.registerBlock(prboxActivePower, "prboxActivePower");
    GameRegistry.registerBlock(timedPRbox, ItemTimedPRbox.class, "timedPRbox");
    GameRegistry.registerBlock(timedPRboxPower, ItemTimedPRbox.class, "timedPRboxPower");
    /* Recipes */
    ItemStack regularPRbox = ItemPRbox.create(-1, 1);
    GameRegistry.addRecipe(regularPRbox, new Object[] {
        " * ", "X X", " * ",
        Character.valueOf('X'), QCraft.Blocks.quantumComputer
    });
    ItemStack timedPRbox = ItemTimedPRbox.create(-1, 1);
    GameRegistry.addRecipe(timedPRbox, new Object[] {
        " * ", "X X", " * ",
        Character.valueOf('X'), prbox, Character.valueOf('Y'), Items.clock
    });
    GameRegistry.addRecipe(new EntangledPRboxRecipe());
    RecipeSorter.register("prbox:entangled_prbox", EntangledPRboxRecipe.class, SHADED, "after:minecraft:shapeless");
    GameRegistry.addRecipe(new EntangledTimedPRboxRecipe());
    RecipeSorter.register("prbox:entangled_timedPRbox", EntangledTimedPRboxRecipe.class, SHADED, "after:minecraft:shapeless");
}

@EventHandler
public void init(FMLInitializationEvent event) {
    /* Register Entities */
    GameRegistry.registerTileEntity(TileEntityPRbox.class, "prbox");
    GameRegistry.registerTileEntity(TileEntityTimedPRbox.class, "timedprbox");
    /* Register ForgeHandler */
    registerForgeHandlers();
}

private void registerForgeHandlers() {
    ForgeHandlers handlers = new ForgeHandlers();
    MinecraftForge.EVENT_BUS.register(handlers);
    FMLCommonHandler.instance().bus().register(handlers);
    ConnectionHandler connectionHandler = new ConnectionHandler();
    MinecraftForge.EVENT_BUS.register(connectionHandler);
    FMLCommonHandler.instance().bus().register(connectionHandler);
}

@EventHandler
public void postInit(FMLPostInitializationEvent event) {
}

private File getWorldSaveLocation(World world, String subPath) {
    File rootDir = FMLCommonHandler.instance().getMinecraftServerInstance().getFile(".");
    File saveDir = null;
    if (QCraft.isServer()) {
        saveDir = new File(rootDir, world.getSaveHandler().getWorldDirectoryName());
    } else {
        saveDir = new File(rootDir, "saves/" + world.getSaveHandler().getWorldDirectoryName());
    }
    return saveDir;
}
```
private File getEntanglementSaveLocation(World world) {
  return getWorldSaveLocation(world, "quantum/entanglements.bin");
}

private File getEncryptionSaveLocation(World world) {
  return getWorldSaveLocation(world, "quantum/encryption.bin");
}

public class ForgeHandlers {
  int count = 0;

  private ForgeHandlers() {
  }
}

// Forge event responses

@SubscribeEvent
public void timer(ServerTickEvent event) {
  count++;
  if (count >= timer) {
    count = 0;
    // System.out.println("------------------------- Timer -------------------------");
    for (int i = 1; i < 1024; i++) {
      List<TileEntityTimedPRbox> list = TileEntityTimedPRbox.TimedPRboxRegistry.getEntangledObjects(i);
      if (list != null) {
        // System.out.println("#EntangledTimedPRboxes: " + list.size());
        if (!list.isEmpty()) {
          Iterator<TileEntityTimedPRbox> it = list.iterator();
          TileEntityTimedPRbox box = it.next();
          World world = box.getWorldObj();
          timedPRbox.powerUpdate(world, box, new Random());
          timedPRbox.setBlockPower(world, list);
        }
      }
    }
  }
}

@SubscribeEvent
public void onWorldLoad(Load event) {
  if (!event.world.isRemote) {
    // Reset
    TileEntityPRbox.PRboxRegistry.reset();
    TileEntityTimedPRbox.TimedPRboxRegistry.reset();
    // Load NBT
    NBTTagCompound rootnbt = loadNBTFromPath(getEntanglementSaveLocation(event.world));
    // Load from NBT
    if (rootnbt != null) {
      if (rootnbt.hasKey("prboxs")) {
        NBTTagCompound prboxs = rootnbt.getCompoundTag("prboxs");
        TileEntityPRbox.PRboxRegistry.readFromNBT(prboxs);
      }
      if (rootnbt.hasKey("timedprboxs")) {
        NBTTagCompound timedprboxs = rootnbt.getCompoundTag("timedprboxs");
        TileEntityTimedPRbox.TimedPRboxRegistry.readFromNBT(timedprboxs);
      }
    }
    // System.out.println("load world...");
  }
}

@SubscribeEvent
public void onWorldUnload(Unload event) {
  if (!event.world.isRemote) {
    // Reset
    TileEntityPRbox.PRboxRegistry.reset();
    TileEntityTimedPRbox.TimedPRboxRegistry.reset();
    //System.out.println("unload world...");
  }
}

@SubscribeEvent
public void onWorldSave(Save event) {
  if (!event.world.isRemote) {
    // Write to NBT
    NBTTagCompound rootnbt = new NBTTagCompound();
    NBTTagCompound prboxs = new NBTTagCompound();
    TileEntityPRbox.PRboxRegistry.writeToNBT(prboxs);
    rootnbt.setTag("prboxs", prboxs);
    //NBTTagCompound timedprboxs = new NBTTagCompound();
    //rootnbt.setTag("timedprboxs", timedprboxs);
  }
}
APPENDIX A. PR-BOX MODIFICATION CODE

```java
NBTTagCompound timedprboxs = new NBTTagCompound();
TileEntityTimedPRbox.TimedPRboxRegistry.writeToNBT(timedprboxs);
rootnbt.setTag("timedprboxs", timedprboxs);
// Save NBT
saveNBTToPath(getEntanglementSaveLocation(event.world), rootnbt);

// System.out.println("save world...");

public static NBTTagCompound loadNBTFromPath(File file) {
    try {
        if (file != null && file.exists()) {
            InputStream input = new BufferedInputStream(new FileInputStream(file));
            try {
                return CompressedStreamTools.readCompressed(input);
            } finally {
                input.close();
            }
        } catch (IOException e) {
            PRbox.log("Warning: failed to load PRbox entanglement info");
        }
        return null;
    }
}

public static void saveNBTToPath(File file, NBTTagCompound nbt) {
    try {
        if (file != null) {
            file.getParentFile().mkdirs();
            OutputStream output = new BufferedOutputStream(new FileOutputStream(file));
            try {
                CompressedStreamTools.writeCompressed(nbt, output);
            } finally {
                output.close();
            }
        } catch (IOException e) {
            PRbox.log("Warning: failed to save PRbox entanglement info");
        }
    }
}

public static void log(String text) {
    System.out.println("[PRbox] " + text);
}
```

**Block Classes**

```java
package tys.prbox;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
import net.minecraft.block.Block;
import net.minecraft.block.BlockRedstoneDiode;
import net.minecraft.block.ITileEntityProvider;
import net.minecraft.client.renderer.texture.IIconRegister;
import net.minecraft.entity.EntityLivingBase;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.item.ItemStack;
import net.minecraft.tileentity.TileEntity;
import net.minecraft.util.ChatComponentTranslation;
import net.minecraft.util.Direction;
import net.minecraft.util.IIcon;
import net.minecraft.util.MathHelper;
import net.minecraft.util.MovingObjectPosition;
import net.minecraft.world.IBlockAccess;
import net.minecraft.world.IBlockEntityData;

public class BlockPRbox extends BlockRedstoneDiode implements ITileEntityProvider {
    // appearance +
    private static class Icons {
        public static final IIcon Front;
        public static final IIcon TopBottom;
        public static final IIcon Side;
    }

    public BlockPRbox(boolean isRepeaterPowered) {
        super(isRepeaterPowered);
    }
}
```
@Override
public Item getItemDropped(int i, Random random, int j) {
    return Item.getItemFromBlock(this);
}

@Override
public Item damageDropped(int i) {
    return null;
}

@Override
public void dropBlockAsItemWithChance(World world, int x, int y, int z, int side, float f, int unknown) {
    // RemoveBlockByPlayer handles this instead
}

@Override
public ArrayList<ItemStack> getDrops(World world, int x, int y, int z, int metadata, int fortune) {
    ArrayList<ItemStack> blocks = new ArrayList<ItemStack>();
    TileEntity entity = world.getTileEntity(x, y, z);
    if (entity != null && entity instanceof TileEntityPRbox) {
        // Get the box back
        TileEntityPRbox box = (TileEntityPRbox) entity;
        ItemStack stack = ItemPRbox.create(box.getEntanglementFrequency(), 1);
        blocks.add(stack);
    }
    return blocks;
}

protected boolean shouldDropItemsInCreative(World world, int x, int y, int z) {
    return false;
}

@Override
public boolean removedByPlayer(World world, EntityPlayer player, int x, int y, int z) {
    if (world.isRemote) {
        return false;
    }
    if (!player.capabilities.isCreativeMode || shouldDropItemsInCreative(world, x, y, z)) {
        // Regular and silk touch block (identical)
        int metadata = world.getBlockMetadata(x, y, z);
        ArrayStack<ItemStack> items = getDrops(world, x, y, z, metadata, 0);
        while (items.size() > 0) {
            ItemStack item = items.get(0);
            dropBlockAsItem(world, x, y, z, item);
        }
    }
    return super.removedByPlayer(world, player, x, y, z);
}

@Override
public void onBlockPlacedBy(World world, int x, int y, int z, EntityLivingBase p_149689_5_, ItemStack p_149689_6_) {
    int l = MathHelper.floor_double((double)(p_149689_5_.rotationYaw * 4.0F / 360.0F) + 2.5D) & 3;
    world.setBlockMetadataWithNotify(x, y, z, l, 2);
    System.out.println("Metadata: " + world.getBlockMetadata(x, y, z));
    updateTick(world, x, y, z, new Random());
}

/* energizing by right-clicking */
@Override
public boolean onBlockActivated(World world, int x, int y, int z, EntityPlayer player, int meta, float x_off, float y_off, float z_off) {
    if (world.isRemote) {
        System.out.println("Find the box and its twins /");
        int l = world.getBlockMetadata(x, y, z);
        TileEntity entity = world.getTileEntity(x, y, z);
        List<TileEntityPRbox> twins = null;
    }
}
APPENDIX A. PR-BOX MODIFICATION CODE

TileEntityPRbox box = null;
if (entity != null && entity instanceof TileEntityPRbox) {
    box = (TileEntityPRbox) entity;
} else {
    System.out.println("WARNING: Why is this not a TileEntityPRbox?");
}

if (box == null) return false;

/* handle the energizing */
Random random = new Random();
powerUpdate(world, box, random);
setBlockPower(world, box);
}
if (world.isRemote) {
    // Inform Player
    player.addChatMessage(new ChatComponentTranslation("msg.prbox_gotenergized"));
}
return true;

/* orientation of the block */
@Override
public IIcon getIcon(IBlockAccess world, int i, int j, int k, int side) {
    if (side == 0 || side == 1) {
        return Icons.TopBottom;
    }
    int metadata = world.getBlockMetadata(i, j, k);
    int direction = Direction.directionToFacing[getDirection(metadata)];
    if (side == direction) {
        return Icons.Front;
    }
    return Icons.Side;
}

@Override
public IIcon getIcon(int side, int damage) {
    switch (side) {
        case 0:
        case 1: {
            return Icons.TopBottom;
        }
        case 4: {
            return Icons.Front;
        }
        default: {
            return Icons.Side;
        }
    }
}

@Override
protected int func_149901_b(int p_149901_1_) {
    return 0;
}

@Override
protected BlockRedstoneDiode getPowered() {
    return PRbox.prboxActivePower;
}

@Override
protected BlockRedstoneDiode getUnpowered() {
    return PRbox.prboxActive;
}

/*@handle changes in power supply */
public boolean powerUpdate(World world, TileEntityPRbox box, Random random) {
    List<TileEntityPRbox> twins = box.findEntangledBoxes();
    TileEntityPRbox last = null;
    boolean inputAND = true;
    boolean outputXOR = false;
    boolean rand;
    if (twins != null) {
        iterator.TileEntityPRbox it = twins.iterator();
        while (it.hasNext()) {
            TileEntityPRbox itbox = it.next();
            rand = random.nextBoolean();
            inputAND &= itbox.getRedstonePowered();
            if (itbox.getEnergized()) {
                outputXOR ^= itbox.getRedstonePowering();
            } else {
                outputXOR ^= rand;
            }
        }
        last = itbox;
    }
if(inputAND != outputXOR && twins.size() > 1) {
    last.setRedstonePowering(!last.getRedstonePowering());
    return true;
}
return false;

public void setBlockPower(World world, TileEntityPRbox box) {
    int frequency = box.getEntanglementFrequency();
    boolean m_powered = box.getRedstonePowered();
    boolean m_powering = box.getRedstonePowering();
    int x = box.xCoord;
    int y = box.yCoord;
    int z = box.zCoord;
    box.unregister();
    if(box.getRedstonePowering()) {
        world.setBlock(x, y, z, this.getBlockPowered(), world.getBlockMetadata(x, y, z), 2);
    } else {
        world.setBlock(x, y, z, this.getBlockUnpowered(), world.getBlockMetadata(x, y, z), 2);
    }
    TileEntityPRbox activeBox = ((TileEntityPRbox) world.getTileEntity(x, y, z));
    activeBox.xCoord = x;
    activeBox.yCoord = y;
    activeBox.zCoord = z;
    activeBox.setEntanglementFrequency(frequency);
    activeBox.setRedstonePowered(m_powered);
    activeBox.setRedstonePowering(m_powering);
    activeBox.setEnergized(true);
}

/* Ticks the block if it's been scheduled */
public void updateTick(World world, int x, int y, int z, Random random) {
    int l = world.getBlockMetadata(x, y, z);
    TileEntity entity = world.getTileEntity(x, y, z);
    TileEntityPRbox box = null;
    if(entity != null && entity instanceof TileEntityPRbox) {
        box = (TileEntityPRbox) entity;
        twins = box.findEntangledBoxes();
    } else {
        System.out.println("WARNING: Why is this not a TileEntityPRbox?");
    }
    if(box == null) return;
    // new state?
    boolean flag = this.isGettingInput(world, x, y, z, l);
    if(box.getRedstonePowered() && !flag) {
        /* unpowered */
        box.setRedstonePowered(false);
    } else if (!box.getRedstonePowered() && flag) {
        /* powered */
        box.setRedstonePowered(true);
    }
    @Override
    /* onNeighborChange -> call updateTick! */
    protected void func_149897_b(World world, int x, int y, int z, Block block) {
        int l = world.getBlockMetadata(x, y, z);
        TileEntity entity = world.getTileEntity(x, y, z);
        TileEntityPRbox box = null;
        if(entity != null && entity instanceof TileEntityPRbox) {
            box = (TileEntityPRbox) entity;
        } else {
            System.out.println("WARNING: Why is this not a TileEntityPRbox?");
        }
        if((box.getRedstonePowered() && !flag || !box.getRedstonePowered() && flag) && !world.isBlockTickScheduledThisTick(x, y, z, this)) {
            byte b0 = -1;
            if(this.func_149912_i(world, x, y, z, l))
                b0 = -3;
        }
    }
APPENDIX A. PR-BOX MODIFICATION CODE

```java
308     } else if (box.getRedstonePowered())
309     {
310         b0 = -2;
311     }
312     world.scheduleBlockUpdateWithPriority(x, y, z, this, this.func_149901_b(l), b0);

313     }

314     @Override
315     public boolean canConnectRedstone(IBlockAccess world, int x, int y, int z, int side) {
316         return true;
317     }

318     /* PR-boxes are TileEntities */
319     @Override
320     public TileEntity createNewTileEntity(World world, int metadata) {
321         return new TileEntityPRbox();
322     }

323     @Override
324     public TileEntity createTileEntity(World world, int metadata) {
325         return createNewTileEntity(world, metadata);
326     }

After the PR-box has been energized, it is represented by the following class.

```java
48
```
case 4: {
    return Icons.Front;
}
default: {
    return Icons.Side;
}
}
/* Called when the block is placed in the world. */
public void onBlockPlacedBy(World world, int x, int y, int z, EntityLivingBase p_149689_5_, ItemStack p_149689_6_) {
    int l = MathHelper.floor_double((double)(p_149689_5_.rotationYaw * 4.0F / 360.0F) + 2.5D) & 3;
    world.setBlockMetadataWithNotify(x, y, z, l, 2);
    //System.out.println("Metadata: " + world.getBlockMetadata(x, y, z));
}
@Override
public void getItemDropped(int i, Random random, int j) {
    return null;
}
@Override
public int damageDropped(int i) {
    return 0;
}
@Override
public ArrayList<ItemStack> getDrops(World world, int x, int y, int z, int metadata, int fortune) {
    ArrayList<ItemStack> blocks = new ArrayList<ItemStack>();
    return blocks;
}
protected boolean shouldDropItemsInCreative(World world, int x, int y, int z) {
    return false;
}
@Override
public boolean removedByPlayer(World world, EntityPlayer player, int x, int y, int z) {
    if (world.isRemote) {
        return false;
    }
    ((TileEntityPRbox) world.getTileEntity(x, y, z)).unregister();
    if (!player.capabilities.isCreativeMode || shouldDropItemsInCreative(world, x, y, z)) {
        int metadata = world.getBlockMetadata(x, y, z);
        ArrayList<ItemStack> items = getDrops(world, x, y, z, metadata, 0);
        Iterator<ItemStack> it = items.iterator();
        while (it.hasNext()) {
            ItemStack item = it.next();
            dropBlockAsItem(world, x, y, z, item);
        }
    }
    return super.removedByPlayer(world, player, x, y, z);
}
@Override
public ItemStack getPickBlock(MovingObjectPosition target, World world, int x, int y, int z) {
    int metadata = world.getBlockMetadata(x, y, z);
    ArrayList<ItemStack> items = getDrops(world, x, y, z, metadata, 0);
    if (items.size() > 0) {
        return items.get(0);
    }
    return null;
}
@Override
public void updateTick(World p_149674_1_, int p_149674_2_, int p_149674_3_, int p_149674_4_, Random p_149674_5_) {
}
@Override
protected void func_149901_b(int p_149901_1_) {
    // TODO Auto-generated method stub
    return 0;
}
APPENDIX A. PR-BOX MODIFICATION CODE

```
@Override
protected BlockRedstoneDiode getBlockPowered() {
    // TODO Auto-generated method stub
    return PRbox.prboxActivePower;
}

@Override
protected BlockRedstoneDiode getBlockUnpowered() {
    // TODO Auto-generated method stub
    return PRbox.prboxActive;
}

/* PR-boxes are TileEntities */
@Override
public TileEntity createNewTileEntity(World world, int metadata) {
    return new TileEntityPRbox();
}

@Override
public TileEntity createTileEntity(World world, int metadata) {
    return createNewTileEntity(world, metadata);
}
```
super.validate();
register();

@Override
public void invalidate() {
    unregister();
    super.invalidate();
}

@Override
public void setEntanglementFrequency(int frequency) {
    if (m_entanglementFrequency != frequency) {
        unregister();
        m_entanglementFrequency = frequency;
        register();
    }
}

@Deprecated
public int getEntanglementFrequency() {
    return m_entanglementFrequency;
}

protected TileEntityPRbox findEntangledTwin() {
    if (m_entanglementFrequency >= 0) {
        List<TileEntityPRbox> twins = PRboxRegistry.getEntangledObjects(m_entanglementFrequency);
        if (twins != null) {
            Iterator<TileEntityPRbox> it = twins.iterator();
            while (it.hasNext()) {
                TileEntityPRbox box = it.next();
                if (box != this) {
                    return box;
                }
            }
        }
    }
    return null;
}

protected List<TileEntityPRbox> findEntangledBoxes() {
    if (m_entanglementFrequency >= 0) {
        return PRboxRegistry.getEntangledObjects(m_entanglementFrequency);
    }
    return null;
}

private void notifyBlockOfNeighborChange(int x, int y, int z) {
    worldObj.notifyBlockOfNeighborChange(x, y, z, worldObj.getBlock(x, y, z));
}

@Override
public void readFromNBT(NBTTagCompound nbttagcompound) {
    // Read properties
    super.readFromNBT(nbttagcompound);
    m_powered = nbttagcompound.getBoolean("p");
    m_powering = nbttagcompound.getBoolean("pa");
    m_entanglementFrequency = nbttagcompound.getInteger("f");
    m_energized = nbttagcompound.getBoolean("en");
}

@Override
public void writeToNBT(NBTTagCompound nbttagcompound) {
    // Write properties
    super.writeToNBT(nbttagcompound);
    nbttagcompound.setBoolean("p", m_powered);
    nbttagcompound.setBoolean("pa", m_powering);
    nbttagcompound.setInteger("f", m_entanglementFrequency);
    nbttagcompound.setBoolean("en", m_energized);
}

@Override
public Packet getDescriptionPacket() {
    // Communicate networking state
    NBTTagCompound nbttagcompound = new NBTTagCompound();
    nbttagcompound.setInteger("f", m_entanglementFrequency);
    return new S35PacketUpdateTileEntity(this.xCoord, this.yCoord, this.zCoord, 0, nbttagcompound);
}
public void onDataPacket(NetworkManager net, S35PacketUpdateTileEntity packet) {
    switch (packet.func_148853_f()) { // actionType
        case 0: { // Read networked state
            NBTTagCompound nbttagcompound = packet.func_148857_g(); // data
            setEntanglementFrequency(nbttagcompound.getInteger("f"));
            break;
        }
        default: {
            break;
        }
    }
}

public void setRedstonePowered(boolean powered) {
    if (m_powered != powered) {
        m_powered = powered;
    }
}

public void setRedstonePowering(boolean powering) {
    if (m_powering != powering) {
        m_powering = powering;
    }
}

public boolean getRedstonePowered() {
    return m_powered;
}

public boolean getRedstonePowering() {
    return m_powering;
}

Item Class

package tys.prbox;
import java.util.List;
import dan200.QCraft;
import dan200.qcraft.shared.ItemQBlock;
import net.minecraft.block.Block;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.item.ItemStack;
import net.minecraft.nbt.NBTTagCompound;
import net.minecraft.tileentity.TileEntity;
import net.minecraft.world.World;

public class ItemPRbox extends ItemBlock {
    public ItemPRbox(Block block) {
        super(block);
        setMaxStackSize(64);
        setHasSubtypes(true);
        setUnlocalizedName("prbox:prbox");
        setCreativeTab(QCraft.getCreativeTab());
    }

    public static ItemStack create(int entanglementFrequency, int quantity) {
        ItemStack result = new ItemStack(PRbox.prbox, quantity, 0);
        setEntanglementFrequency(result, entanglementFrequency);
        return result;
    }

    @Override
    public void getSubItems(Item item, CreativeTabs tabs, List list) {
        list.add(create(-1, 1));
    }

    public static void setEntanglementFrequency(ItemStack stack, int entanglementFrequency) {
        // Ensure the nbt
        if (!stack.hasTagCompound()) {
            stack.setTagCompound(new NBTTagCompound());
        }
        // Set the tags
        NBTTagCompound nbt = stack.getTagCompound();
        if (entanglementFrequency < 0) {
            // No frequency
            if (nbt.hasKey("e")) {
                nbt.removeTag("e");
            }
            if (nbt.hasKey("R")) {
                nbt.removeTag("R");
            }
        } else {
            stack.getTagCompound().setInteger("f", entanglementFrequency);
        }
        stack.setTagCompound(nbt);
    }
else if (entanglementFrequency == 0) {
    // Unknown frequency
    nbt.setInteger("e", entanglementFrequency);
} else {
    // Known frequency
    nbt.setInteger("e", entanglementFrequency);
    if (nbt.hasKey("R")) {
        nbt.removeTag("R");
    }
}
}

public static int getEntanglementFrequency(ItemStack stack) {
    if (stack.hasTagCompound()) {
        NBTTagCompound nbt = stack.getTagCompound();
        if (nbt.hasKey("e")) {
            int frequency = nbt.getInteger("e");
            return frequency;
        }
    }
    return -1;
}

@Override
public void onCreated(ItemStack stack, World world, EntityPlayer player) {
    if (getEntanglementFrequency(stack) == 0 && !world.isRemote) {
        setEntanglementFrequency(stack, TileEntityPRbox.getEntanglementRegistry(world).getUnusedFrequency());
        player.inventory.markDirty();
        if (player.openContainer != null) {
            player.openContainer.detectAndSendChanges();
        }
    }
}

@Override
public boolean placeBlockAt(ItemStack stack, EntityPlayer player, World world, int x, int y, int z, int side, float hitX, float hitY, float hitZ, int metadata) {
    if (super.placeBlockAt(stack, player, world, x, y, z, side, hitX, hitY, hitZ, metadata)) {
        TileEntity entity = world.getTileEntity(x, y, z);
        if (entity != null && entity instanceof TileEntityPRbox) {
            TileEntityPRbox quantum = (TileEntityPRbox) entity;
            quantum.setEntanglementFrequency(getEntanglementFrequency(stack));
        }
        return true;
    }
    return false;
}

@Override
public String getUnlocalizedName(ItemStack stack) {
    if (getEntanglementFrequency(stack) >= 0) {
        return "tile.prbox:prbox_entangled";
    }
    return "tile.prbox:prbox";
}

@Override
public void addInformation(ItemStack stack, EntityPlayer player, List list, boolean par4) {
    int frequency = getEntanglementFrequency(stack);
    if (frequency > 0) {
        list.add("Group: " + ItemQBlock.formatFrequency(frequency));
    }
}

package tys.prbox;

import dan200.QCraft;
import dan200.qcraft.shared.ItemEOS;
import net.minecraft.inventory.InventoryCrafting;
import net.minecraft.item.ItemStack;
import net.minecraft.item.crafting.IRecipe;
import net.minecraft.world.World;

public class EntangledPRboxRecipe implements IRecipe {
    public EntangledPRboxRecipe() {
    }

    @Override
    public int getRecipeSize() {
        return 9;
    }

    public String getUnlocalizedName() {
        return "tile.prbox:prbox_entangled";
    }

    @Override
    public void addInformation(ItemStack stack, EntityPlayer player, List list, boolean par4) {
        int frequency = getEntanglementFrequency(stack);
        if (frequency > 0) {
            list.add("Group: " + ItemQBlock.formatFrequency(frequency));
        }
    }

    @Override
    public boolean placeBlockAt(ItemStack stack, EntityPlayer player, World world, int x, int y, int z, int side, float hitX, float hitY, float hitZ, int metadata) {
        if (super.placeBlockAt(stack, player, world, x, y, z, side, hitX, hitY, hitZ, metadata)) {
            TileEntity entity = world.getTileEntity(x, y, z);
            if (entity != null && entity instanceof TileEntityPRbox) {
                TileEntityPRbox quantum = (TileEntityPRbox) entity;
                quantum.setEntanglementFrequency(getEntanglementFrequency(stack));
            }
            return true;
        }
        return false;
    }

    @Override
    public String getUnlocalizedName(ItemStack stack) {
        if (getEntanglementFrequency(stack) >= 0) {
            return "tile.prbox:prbox_entangled";
        }
        return "tile.prbox:prbox";
    }

    @Override
    public void addInformation(ItemStack stack, EntityPlayer player, List list, boolean par4) {
        int frequency = getEntanglementFrequency(stack);
        if (frequency > 0) {
            list.add("Group: " + ItemQBlock.formatFrequency(frequency));
        }
    }

    @Override
    public int getRecipeSize() {
        return 9;
    }
@Override
public ItemStack getRecipeOutput() {
    return ItemPRbox.create(0, 2);
}

@Override
public boolean matches(InventoryCrafting _inventory, World world) {
    return (getCraftingResult(_inventory) != null);
}

@Override
public ItemStack getCraftingResult(InventoryCrafting inventory) {
    // Find the eos
    int eosPosX = -1;
    int eosPosY = -1;
    for (int y = 0; y < 3; ++y) {
        for (int x = 0; x < 3; ++x) {
            ItemStack item = inventory.getStackInRowAndColumn(x, y);
            if (item != null && item.getItem() == QCraft.Items.eos && item.getItemDamage() == ItemEOS.SubType.Entanglement) {
                eosPosX = x;
                eosPosY = y;
                break;
            }
        }
    }
    // Fail if no eos found:
    if (eosPosX < 0 || eosPosY < 0) {
        return null;
    }
    // Find box
    int boxsFound = 0;
    int entangled = 0;
    int frequency = 0;
    for (int y = 0; y < 3; ++y) {
        for (int x = 0; x < 3; ++x) {
            if (!(x == eosPosX && y == eosPosY)) {
                if ((x == eosPosX - 1 || x == eosPosX + 1) && y == eosPosY) {
                    // Find box (must be unentangled)
                    ItemStack odb = inventory.getStackInRowAndColumn(x, y);
                    if (odb != null && odb.getItem() instanceof ItemPRbox) {
                        if (ItemPRbox.getEntanglementFrequency(odb) > 0) {
                            entangled++;
                            if (entangled > 1) {
                                return null;
                            }
                            boxsFound++;
                        } else {
                            // Ensure empty
                            if (inventory.getStackInRowAndColumn(x, y) != null) {
                                return null;
                            }
                        }
                    } else {
                        // Ensure empty
                        if (inventory.getStackInRowAndColumn(x, y) != null) {
                            return null;
                        }
                    }
                } else {
                    // Ensure empty
                    if (inventory.getStackInRowAndColumn(x, y) != null) {
                        return null;
                    }
                }
            }
        }
    }
    // Check box is found
    if (boxsFound != 2) {
        return null;
    }
    // Determine frequency
    int entanglementFrequency = frequency; // Will be filled in after crafting
    // Create item
    return ItemPRbox.create(entanglementFrequency, boxsFound);
}

TimedPRbox

Block Class

package tys.prbox;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import net.minecraft.block.Block;
import net.minecraft.block.BlockRedstoneDiode;
import net.minecraft.block.ITileEntityProvider;
import net.minecraft.client.renderer.texture.IIconRegister;
import net.minecraft.entity.EntityLivingBase;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.item.ItemStack;
import net.minecraft.tileentity.TileEntity;
import net.minecraft.util.Direction;
import net.minecraft.util.IIcon;
import net.minecraft.util.MathHelper;
import net.minecraft.util.MovingObjectPosition;
import net.minecraft.world.IBlockAccess;
import net.minecraft.world.World;

public class BlockTimedPRbox extends BlockRedstoneDiode implements ITileEntityProvider {

  private static class Icons {
    public static IIcon Front;
    public static IIcon TopBottom;
    public static IIcon Side;
  }

  public BlockTimedPRbox(boolean isRepeaterPowered) {
    super(isRepeaterPowered);
  }

  @Override
  public Item getItemDropped(int i, Random random, int j) {
    return Item.getItemFromBlock(this);
  }

  @Override
  public int damageDropped(int i) {
    return 0;
  }

  @Override
  public void dropBlockAsItemWithChance(World world, int x, int y, int z, int side, float f, int unknown) {
    // RemoveBlockByPlayer handles this instead
  }

  @Override
  public ArrayList<ItemStack> getDrops(World world, int x, int y, int z, int metadata, int fortune) {
    ArrayList<ItemStack> blocks = new ArrayList<ItemStack>();
    TileEntity entity = world.getTileEntity(x, y, z);
    if (entity != null && entity instanceof TileEntityTimedPRbox) {
      // Get the box back
      TileEntityTimedPRbox box = (TileEntityTimedPRbox) entity;
      ItemStack stack = ItemTimedPRbox.create(box.getEntanglementFrequency(), 1);
      blocks.add(stack);
    }
    return blocks;
  }

  protected boolean shouldDropItemsInCreative(World world, int x, int y, int z) {
    return false;
  }

  @Override
  public boolean removedByPlayer(World world, EntityPlayer player, int x, int y, int z) {
    if (world.isRemote) {
      return false;
    }
    if (!player.capabilities.isCreativeMode || shouldDropItemsInCreative(world, x, y, z)) {
      // Regular and silk touch block (identical)
      metadata = world.getBlockMetadata(x, y, z);
      ArrayList<ItemStack> items = getDrops(world, x, y, z, metadata, 0);
      Iterator<ItemStack> it = items.iterator();
      while (it.hasNext()) {
        ItemStack item = it.next();
        dropBlockAsItem(world, x, y, z, item);
      }
    }
    return super.removedByPlayer(world, player, x, y, z);
  }

  @Override
  public ItemStack getPickBlock(MovingObjectPosition target, World world, int x, int y, int z) {
    return super.getPickBlock(target, world, x, y, z);
  }
}
ArrayList<ItemStack> items = getDrops(world, x, y, z, metadata, 0);
if(items.size() > 0) {
    return items.get(0);
} else {
    return null;
}

@Override
public void registerBlockIcons(IIconRegister iconRegister) {
    Icons.Front = iconRegister.registerIcon("prbox:time@prbox_input");
    Icons.TopBottom = iconRegister.registerIcon("prbox:timedprbox_topbottom");
    Icons.Side = iconRegister.registerIcon("prbox:timedprbox_output");
}

/**
 * Called when the block is placed in the world.
 */
public void onBlockPlacedBy(World world, int x, int y, int z, EntityLivingBase p_149689_5_, ItemStack p_149689_6_) {
    int l = MathHelper.floor_double((double)(p_149689_5_.rotationYaw * 4.0F / 360.0F) + 2.5D) & 3;
    world.setBlockMetadataWithNotify(x, y, z, l, 2);
    System.out.println("Metadata: " + world.getBlockMetadata(x, y, z));
}

@Override
public IIcon getIcon(IBlockAccess world, int i, int j, int k, int side) {
    if(side == 0 || side == 1) {
        return Icons.TopBottom;
    } else if(side == directionToFacing(getDirection(metadata))) {
        return Icons.Front;
    } else {
        return Icons.Side;
    }
}

@Override
public IIcon getIcon(int side, int damage) {
    switch(side) {
    case 0: {
        case 1: {
            return Icons.TopBottom;
        }
         case 4: {
            return Icons.Front;
        }
        default: {
            return Icons.Side;
        }
    };
}

/*------------*/
/* Diode-Code */
/*------------*/

@Override
protected int func_149901_b(int p_149901_1_) {
    return 0;
}

@Override
protected BlockRedstoneDiode getBlockPowered() {
    return PRbox.timedPRboxPower;
}

@Override
protected BlockRedstoneDiode getBlockUnpowered() {
    return PRbox.timedPRbox;
}

public boolean powerUpdate(World world, TileEntityTimedPRbox box, Random random) {
    List<TileEntityTimedPRbox> twins = box.findEntangledBoxes();
    TileEntityTimedPRbox last = null;
    boolean outputXOR = false;
    boolean rand = false;
    if(twins != null) {
        Iterator<TileEntityTimedPRbox> it = twins.iterator();
        while(it.hasNext()) {
            TileEntityTimedPRbox ithbox = it.next();
            rand = random.nextBoolean();
            if(rand) {
                outputXOR = !outputXOR;
            }
        }
    }
    return outputXOR;
}
inputAND &= itbox.getRedstonePowered();
outputXOR ^= rand;
System.out.println("* + itbox.xCoord + *, y: * + itbox.yCoord + *, z: * + itbox.zCoord + *, power: * + rand + *, getpowered: * + itbox.getRedstonePowered();
last = itbox;
if(inputAND != outputXOR && twins.size() > 1) {
System.out.println("Correct -> x: * + last.xCoord + *, y: * + last.yCoord + *, z: * + last.zCoord + *, power: * + last.getRedstonePowered();
return true;
}
return false;
}
public void setBlockPower(World world, List<TileEntityTimedPRbox> boxes) {
List<TileEntityTimedPRbox> clone = new LinkedList<TileEntityTimedPRbox>();
for(int i = 0; i < boxes.size(); i++) {
clone.add(boxes.get(i));
}
Iterator<TileEntityTimedPRbox> it = clone.iterator();
while(it.hasNext()) {
setBlockPower(world, it.next());
}
}
public void setBlockPower(World world, TileEntityTimedPRbox box) {
int frequency = box.getEntanglementFrequency();
boolean m_powered = box.getRedstonePowered();
boolean m_powering = box.getRedstonePowering();
int x = box.xCoord;
int y = box.yCoord;
int z = box.zCoord;
box.unregister();
System.out.println("SetPower --- x: * + x + *, y: * + y + *, z: * + z + *, power: * + m_powering!;"
if(box.getRedstonePowering()) {
world.setBlock(x, y, z,
this.getBlockPowered(), world.getBlockMetadata(x, y, z), 2);
} else {
world.setBlock(x, y, z,
this.getBlockUnpowered(), world.getBlockMetadata(x, y, z), 2);
}
TileEntity entity = world.getTileEntity(x, y, z);
//if(entity != null) System.out.println("instanceof TileEntityTimedPRbox? " + (entity instanceof TileEntityTimedPRbox));
if(entity != null && entity instanceof TileEntityTimedPRbox) {
box = (TileEntityTimedPRbox) entity;
box.register();
box.setEntanglementFrequency(frequency);
box.setRedstonePowered(m_powered);
box.setRedstonePowering(m_powering);
box.setUpdate = false;
}
else {
System.out.println("WARNING: Why is this not a TileEntityTimedPRbox?
" + (entity instanceof TileEntityTimedPRbox));
}
}
/*
 * Ticks the block if it's been scheduled
 */
public void updateTick(World world, int x, int y, int z, Random random) {
int l = world.getBlockMetadata(x, y, z);
TileEntity entity = world.getTileEntity(x, y, z);
List<TileEntityTimedPRbox> twins = null;
boolean change = false;
TileEntityTimedPRbox box = null;
if(entity != null && entity instanceof TileEntityTimedPRbox) {
box = (TileEntityTimedPRbox) entity;
twins = box.findEntangledBoxes();
} else {
System.out.println("WARNING: Why is this not a TileEntityTimedPRbox?
" + (entity instanceof TileEntityTimedPRbox));
}
if(box == null) return;
// new state?
boolean flag = this.isGettingInput(world, x, y, z, l);
APPENDIX A. PR-BOX MODIFICATION CODE

```java
//System.out.println("getting input? " + flag + ", id: " + box);
if (box.getRedstonePowered() && !flag) {
    /* unpowered */
    box.setRedstonePowered(false);
    change = true;
} else if (!box.getRedstonePowered() && flag) {
    /* powered */
    box.setRedstonePowered(true);
    change = true;
}
//System.out.println("secUpdate: " + box.secUpdate + ", change: " + change);
if (change && box.secUpdate) {
    if (twins != null) {
        Iterator<TileEntityTimedPRbox> it = twins.iterator();
        TileEntityTimedPRbox itbox;
        boolean inputAND = true;
        boolean outputXOR = false;
        while (it.hasNext()) {
            itbox = it.next();
            inputAND &= itbox.getRedstonePowered();
            outputXOR ^= itbox.getRedstonePowering();
        }
        if (inputAND != outputXOR) {
            box.setRedstonePowering(!box.getRedstonePowering());
        }
    }
}
@Override
/* onNeighborChange -> call updateTick! */
protected void func_149897_b(World world, int x, int y, int z, Block block) {
    int l = world.getBlockMetadata(x, y, z);
    boolean flag = this.isGettingInput(world, x, y, z, l);
    TileEntity entity = world.getTileEntity(x, y, z);
    TileEntityTimedPRbox box = null;
    if (entity != null && entity instanceof TileEntityTimedPRbox) {
        box = (TileEntityTimedPRbox) entity;
        System.out.println("WARNING: Why is this not a TileEntityTimedPRbox? 
---------------------------------------------------------");
    }
    if ((box.getRedstonePowered() && !flag || !box.getRedstonePowered() && flag) && !world.isBlockTickScheduledThisTick(x,
        y, z, this)) {
        byte b0 = -1;
        if (this.func_149912_i(world, x, y, z, l)) {
            b0 = -3;
        } else if (box.getRedstonePowered()) {
            b0 = -2;
        }
        world.scheduleBlockUpdateWithPriority(x, y, z, this, this.func_149901_b(l), b0);
    }
}@
@Override
public boolean canConnectRedstone(IBlockAccess world, int x, int y, int z, int side) {
    return true;
}@
@Override
public TileEntity createNewTileEntity(World world, int metadata) {
    return new TileEntityTimedPRbox();
}@
@Override
public TileEntity createTileEntity(World world, int metadata) {
    return createNewTileEntity(world, metadata);
}@
```

**TileEntity Class**

```java
package tys.prbox;
import java.util.Iterator;
import java.util.List;
import dan200.qcraft.shared.EntanglementRegistry;
import net.minecraft.nbt.NBTTagCompound;
import net.minecraft.network.NetworkManager;
import net.minecraft.network.Packet;
import net.minecraft.network.play.server.S35PacketUpdateTileEntity;
import net.minecraft.tileentity.TileEntity;
import net.minecraft.world.World;
```
public class TileEntityTimedPRbox extends TileEntity {
    public static final EntanglementRegistry<TileEntityTimedPRbox> TimedPRboxRegistry = new EntanglementRegistry<TileEntityTimedPRbox>();
    public static final EntanglementRegistry<TileEntityTimedPRbox> ClientTimedPRboxRegistry = new EntanglementRegistry<TileEntityTimedPRbox>();
    public static final int RANGE = 8;

    public static EntanglementRegistry<TileEntityTimedPRbox> getEntanglementRegistry(World world) {
        if (!world.isRemote) {
            return TimedPRboxRegistry;
        } else {
            return ClientTimedPRboxRegistry;
        }
    }

    // Shared state
    private boolean m_powered;
    private boolean m_powering;
    public boolean secUpdate = false;
    private int m_entanglementFrequency;

    public TileEntityTimedPRbox() {
        m_powered = false;
        m_powering = false;
        m_entanglementFrequency = -1;
    }

    public EntanglementRegistry<TileEntityTimedPRbox> getEntanglementRegistry() {
        return getEntanglementRegistry(worldObj);
    }

    @Override
    public void validate() {
        super.validate();
        register();
    }

    @Override
    public void invalidate() {
        unregister();
        super.invalidate();
    }

    // Entanglement
    protected void register() {
        if (m_entanglementFrequency >= 0) {
            getEntanglementRegistry().register(m_entanglementFrequency, this);
        }
    }

    protected void unregister() {
        if (m_entanglementFrequency >= 0) {
            getEntanglementRegistry().unregister(m_entanglementFrequency, this);
        }
    }

    public void setEntanglementFrequency(int frequency) {
        if (m_entanglementFrequency != frequency) {
            unregister();
            m_entanglementFrequency = frequency;
            register();
        }
    }

    public int getEntanglementFrequency() {
        return m_entanglementFrequency;
    }

    protected TileEntityTimedPRbox findEntangledTwin() {
        if (m_entanglementFrequency >= 0) {
            List<TileEntityTimedPRbox> twins = TimedPRboxRegistry.getEntangledObjects(m_entanglementFrequency);
            if (twins != null) {
                Iterator<TileEntityTimedPRbox> it = twins.iterator();
                while (it.hasNext()) {
                    TileEntityTimedPRbox box = it.next();
                    if (box != this) {
                        return box;
                    }
                }
            }
        }
        return null;
    }

    protected List<TileEntityTimedPRbox> findEntangledBoxes() {
        if (m_entanglementFrequency >= 0) {
            return TimedPRboxRegistry.getEntangledObjects(m_entanglementFrequency);
        }
        return null;
    }
private void notifyBlockOfNeighborChange(int x, int y, int z) {
    worldObj.notifyBlockOfNeighborChange(x, y, z, worldObj.getBlock(x, y, z));
}

@Override
public void readFromNBT(NBTTagCompound nbttagcompound) {
    // Read properties
    super.readFromNBT(nbttagcompound);
    m_powered = nbttagcompound.getBoolean("p");
    m_powering = nbttagcompound.getBoolean("pa");
    m_entanglementFrequency = nbttagcompound.getInteger("f");
}

@Override
public void writeToNBT(NBTTagCompound nbttagcompound) {
    // Write properties
    super.writeToNBT(nbttagcompound);
    nbttagcompound.setBoolean("p", m_powered);
    nbttagcompound.setBoolean("pa", m_powering);
    nbttagcompound.setInteger("f", m_entanglementFrequency);
}

@Override
public Packet getDescriptionPacket() {
    // Communicate networked state
    NBTTagCompound nbttagcompound = new NBTTagCompound();
    nbttagcompound.setInteger("f", m_entanglementFrequency);
    return new S35PacketUpdateTileEntity(this.xCoord, this.yCoord, this.zCoord, 0, nbttagcompound);
}

@Override
public void onDataPacket(NetworkManager net, S35PacketUpdateTileEntity packet) {
    switch (packet.func_148853_f()) { // actionType
        case 0:
            // Read networked state
            NBTTagCompound nbttagcompound = packet.func_148857_g(); // data
            setEntanglementFrequency(nbttagcompound.getInteger("f"));
            break;
        default:
            break;
    }
}

public void setRedstonePowered(boolean powered) {
    if (m_powered != powered) {
        m_powered = powered;
    }
}

public void setRedstonePowering(boolean powering) {
    if (m_powering != powering) {
        m_powering = powering;
    }
}

public boolean getRedstonePowered() {
    return m_powered;
}

public boolean getRedstonePowering() {
    return m_powering;
}

Item Class
package tys.prbox;

import java.util.List;
import dan200.QCraft;
import dan200.qcraft.shared.ItemQBlock;
import dan200.qcraft.shared.TileEntityQBlock;
import net.minecraft.block.Block;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.item.ItemStack;
import net.minecraft.nbt.NBTTagCompound;
import net.minecraft.tileentity.TileEntity;
import net.minecraft.world.World;

public class ItemTimedPRbox extends ItemBlock {
    public ItemTimedPRbox(Block block) {
public static ItemStack create(int entanglementFrequency, int quantity) {
    ItemStack result = new ItemStack(PRbox.timedPRbox, quantity, 0);
    setEntanglementFrequency(result, entanglementFrequency);
    return result;
}

public static void getSubItems(Item item, CreativeTabs tabs, List list) {
    list.add(create(-1, 1));
}

public static void setEntanglementFrequency(ItemStack stack, int entanglementFrequency) {
    NBTTagCompound nbt = stack.getTagCompound();
    if (nbt.hasKey("e")) {
        nbt.removeTag("e");
    }
    if (nbt.hasKey("R")) {
        nbt.removeTag("R");
    }
    else if (entanglementFrequency < 0) {
        // No frequency
        nbt.setInteger("e", 0);
        nbt.setInteger("R", TileEntityTimedPRbox.getEntanglementRegistry(stack.getWorld()).getUnusedFrequency());
    }
    else if (entanglementFrequency == 0) {
        // Unknown frequency
        nbt.setInteger("e", 0);
        nbt.setInteger("R", TileEntityTimedPRbox.x_random.nextInt(0xffffff));
    }
    else {  // Known frequency
        nbt.setInteger("e", entanglementFrequency);
        if (nbt.hasKey("R")) {
            nbt.removeTag("R");
        }
    }
}

public static int getEntanglementFrequency(ItemStack stack) {
    if (stack.hasTagCompound()) {
        NBTTagCompound nbt = stack.getTagCompound();
        if (nbt.hasKey("e")) {
            int frequency = nbt.getInteger("e");
            return frequency;
        }
    }
    return -1;
}

@Override
public void onCreated(ItemStack stack, EntityPlayer player, World world, EntityPlayer player, World world, int x, int y, int z, int side, float hitX, float hitY, float hitZ, int metadata) {
    if (super.placeBlockAt(stack, player, world, x, y, z, side, hitX, hitY, hitZ, metadata)) {
        TileEntity entity = world.getTileEntity(x, y, z);
        if (entity instanceof TileEntityTimedPRbox) {
            TileEntityTimedPRbox quantum = (TileEntityTimedPRbox) entity;
            quantum.setEntanglementFrequency(getEntanglementFrequency(stack));
            return true;
        }
    }
    return false;
}

@Override
public void addInformation(ItemStack stack, EntityPlayer player, List list, boolean par4) {
    int frequency = getEntanglementFrequency(stack);
    if (frequency > 0) {
        list.add("Group: " + ItemQBlock.formatFrequency(frequency));
    }
APPENDIX A. PR-BOX MODIFICATION CODE

Correlation Recipe

The correlation recipe of the TimedPRbox is exactly the same as the original one besides exchanging PRbox and TimedPRbox.

Timer Command

```java
class CommandTimer extends CommandBase {
    @Override
    public String getCommandName() {
        return "ptimer";
    }
    @Override
    public String getCommandUsage(ICommandSender p_71518_1_) {
        return "command.timer";
    }
    @Override
    public void processCommand(ICommandSender sender, String[] args) {
        if(sender instanceof EntityPlayer && args[0] != null) {
            PRbox.timer = Integer.parseInt(args[0]);
        }
    }
}
```
Eidesstattliche Versicherung

___________________________   ___________________________
Name, Vorname     Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als
die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf
einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische
Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner
Prüfungsbehörde vorgelegen.

___________________________    ___________________________
Ort, Datum       Unterschrift

* Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung
falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei
Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so
tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158
Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

___________________________    ___________________________
Ort, Datum       Unterschrift