MoneyHeistMachine

Product Report - PROG4 assignment

Richard Kroesen (663339)
Date: 19-6-2022 (dd-mm-yy)
Group: ESE1-A
Lecturer: Peter Bijl
Version: v2.1

Versionlog

Date (mm-dd-yy)	Version	Comment
4-29-2022	V0.1	Concept initialized: layout and table of contents
5-16-2022	V0.2	Requirements composed.
5-19-2022	V0.3	Design phase expanded: state chart, architecture and limits.
6-6-2022	V1.0	Definition of Acceptance test done
6-12-2022	V1.1	Copywriting improvement
6-19-2022	V1.2	Realisation plan writing and testplan
6-19-2022	V2.0	Rounding of the document
6-19-2022	V2.1	Definite product report

Summary

MoneyHeistMachine™ is a slot machine which is developed by the student Richard Kroesen. The purpose of the MoneyHeistMachine is to provide a game for entertainment with a chance to win money for the users. The target market for this product are businesses in the casino sector or nightlife entertainment oriented.

The most important requirements of the applications are: user-friendly graphical interface; use of hardware abstraction; and system stability.

This documentation starts with introduction in chapter one, which gives project background and objectives. Functional, technical and nonfunctional requirements are defined in chapter two. And the global system function described in this section. In chapter three you can find the design of the system, this is documented with the help of state and architecture layers diagrams. Besides the acceptance test report is here defined. The realization of the system can be seen in the fourth chapter, the user interface explanation and program implementation can be read here. For the results of the acceptance test please refer to chapter five. Lastly the conclusions are drawn in chapter six.

Preface

For starters I would like to say this project was informative as an introduction to object-oriented programming with programming language C++. Besides, I have challenged myself to document this project in English and I am glad I did it.

I liked this project a lot, because of the successful version for PROG2. Which was rated as an original, reliable and amusing finite state machine system. For PROG4 I could expand my concept and make the user interface graphical.

Even tough I have implemented every requirement; I still would want to expand this project to a more advanced graphical interaction application.

With this project I learned the following subjects: working with Qt Widgets; programming with OOP in C++; writing technical documentation in English and development of graphical user interfaces.

Lastly, I want to thank my lecturer Peter Bijl for the PROG4 lessons.

Table of Contents

1. Introduction

1.1 Background

The project's MoneyHeistMachine concept stems from previous finite state machine programming assignment for PROG2 class. MoneyHeistMachine is based on a slot machine concept, the famously known gambling machines at the casinos.

The host or the buyer of the machine is mostly a businessperson who places the machine in a nightlife environment, like a casino. The host profits from the amount games played and their stake price. The game is based on randomized chance combinations. Based on thoughtful statistical math model the host is guaranteed to make money, at least eventually.

The users of the slot machine are nightlife consumers or people who wants to take their chances. The purpose of the machine is to play a game for fun with a chance to win money for the users. However, there may be extreme users that are addicted to gambling. Either way, these users risk their money in the hope to win more. Mostly the jackpot payout is very extreme, which tempts people to keep playing.

1.2 Scope of Development

The focus in this project lifecycle is to learn C++ on behave of developing a GUI with Qt widgets library. This project does not have a real client or a detailed assignment. Instead, the developer (student) is free to setup his own requirements and implement them. Note this project has some boundary condition to pass for PROG4 class. These are picked up in the requirements list.

1.3 Main objectives

In this paragraph the primary goals are defined. These are the most important within the scope of this project, but not limited to. See these goals as global and thread for this project.

- Learning the C++ programminglanguage.
- Learning Qt Widgets library by application.
- Creating a graphical user interface to simulate hardware.
- Developing according to Hardware Abstraction Layer (HAL) principle.

2. Definition phase

The definition of the MoneyHeistMachine project is described in this chapter. You can find the main function and the requirements for the product development.

2.1 Product Main Function

The main goal of MoneyHeistMachine™ graphical user interface (GUI) is to supply a hardware application layer (HAL) for the product system. The software of this system will be developed and tested for the client without any hardware engineering involvement.

The product's high-level function will be explained on the basis of a simple input-process-output diagram, see Figure 1.

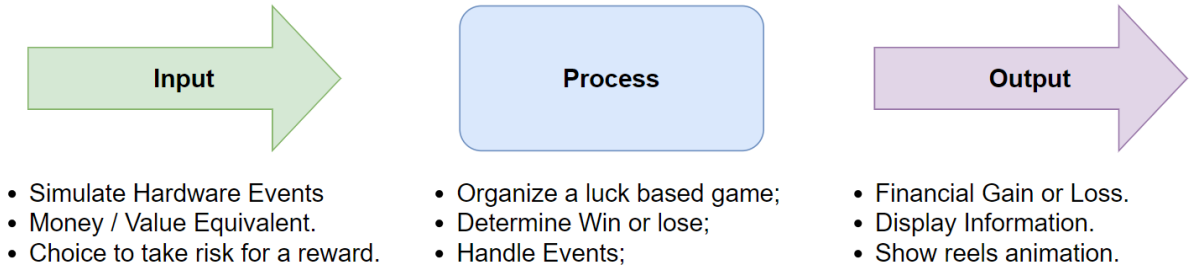

Figure 1 high-level IPO system diagram

The main inputs of the system are hardware events which are simulated by the software interface.

The system process aim is to handle events and manage the gamble game.

Lastly the system outputs information with graphics and text to the user. In a factual system the system will also control various kinds of actuators. Think of the reels with different symbols, tokens or money dispensers, audio effects or eye-catching lighting effects. However, this is outside the scope of this project.

2.2 Requirements

Functional requirements describe the product features, so what the product will do. Technical requirements are the products development constraints or rules. Functional requirements you can find in section 2.2.1. Technical requirements are written you can find at 2.2.2. Besides this project has nonfunctional requirements, which can be found at 2.2.3.

The primary product functions are first order numbered; secondary product functions are nested within them. Which reductions the primary function in smaller pieces. This is done to break a bigger problem into smaller ones, which are easier to implement and to test.

So, for example the primary function of F1 is: "provide access to system's sensitive information". The secondary function of the primary function could be F1.1 logging system.

2.2.1 Functional Requirements

ID	Title	Statement	Comments	Priority
F1	Admin mode	The program shall provide access to system's sensitive information.	Admin mode is intended for device's admins or service engineers.	/
F1.1	Logging system	The program shall have a logging system, which records machine system's states and events.	Use QmessageLogger class.	Must
F1.2	Fraud monitor system	The program shall have a logging system specific for transactions to detect fraud activities.		Wish
F2	User Interface	The system shall provide the user with multiple input possibilities to simulate the hardware with software. (HAL principle) And the system should display relavant information.		Must
F2.1	Insert Money	The system shall accept variable amounts of money, with a precision of integers.		Must
F2.2	Active Spin	The system shall gather an event for triggering a gamble spin.	Without any balance, user shouldn't be able to an active spin.	Must
F2.3	Show Reels (animated)	The system shall show the selection process of the reels combination, during gambling.	The GUI application will need some graphical component for this.	Wish
F2.4	Show Payout	The system shall show the payout table of potential rewards.		Prefered
F2.5	Pay Money	The system shall give the user the ability to get paid his credit.		Must
F2.6	Balance View	The system shall report the current balance amount to the user.		Must
F2.7	Show Reels	The system shall show the output of the reels combination, after gambling.		Must
F2.8	Recovery Option	The program shall have a recovery option for wrong user input.	For this system this means the user can get a full refund without gambling.	Must
F3	Random Reel Combination Generator	The system shall generate a random reel combination, which is adaptive for change.		/
F3.1	Reel Combination generator	The system shall generate a random reel combination with integers.	Actually, the system will be generating pseudo random combination.	Must
F3.2	Reel Generator configuration	The system shall provide a future developer an option/interface to adapt reel generator.		Prefered
F3.3	Payout calculator	The system shall figure out the payout factor based on users bet stake and reel combination.		Must

F3.4	Reel symbol linker	The system shall link the combination numbers to delightful icons/symbols.		Prefered

Table 1 functional requirement

Technical requirements provide a guideline that should be followed during the development life cycle. Think of matters like tools, resources or technical norm.

2.2.2 Technical

ID	Statement	Comments	Priority
T1	Program can be started up and does not crash while in use.		Must
T2	GUI widgets are preferred programmed with C++ and not with QML.		Wish
T3	The program is made with Qt Creator IDE.		Preferred
T4	Hardware event triggers are simulated by the user with the application.		Must
T5	The system is built according to the finite state machine model.		Must
T6	The system is built according to finite state machine principle		Must
T7	The system is meant to be build with hardware abstraction.	The system must work with events that are triggered by software, instead of hardware signals.	Must

Table 2 technical requirements

Nonfunctional requirements define how the product should be implemented or interpreted. These requirements are tough to verify and confirm, because they are subjective. However, it is important to keep them in mind and try to endeavor them.

2.2.3 Nonfunctional

ID	Statement	Comments	Priority
N1	User-friendliness for professional environment.		-
N2	Make a sketch of the GUI with proper widgets. Choose a suitable layout structure.		Must
N3	Program code is readable and informative.	Comments which explain why this line of code is written.	-

Table 3 nonfunctional requirements

3. Design Phase

This chapter describes how the system is going to convert the specification into a technical solution, also known as the processing side of the project. You can find here the description for the global overview of the design.

3.1 State Chart

MoneyHeistMachine system is build according to the finite state machine model. To represent the system behavior within certain states a state chart is created. Which you can see below this paragraph in Figure 2.

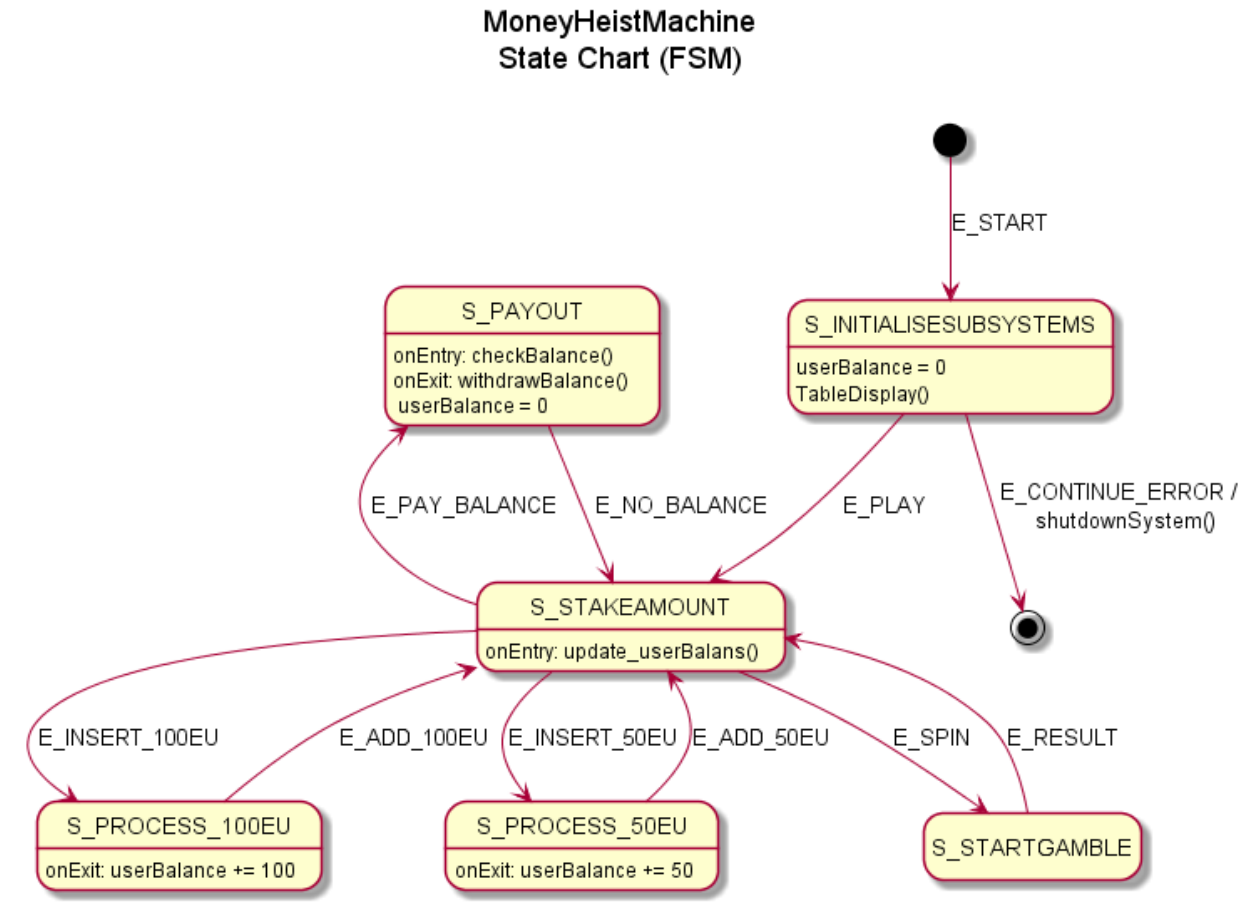

Figure 2 State chart

In the state chart the connection between states is clearly visible. Besides these relationships there could also labels be found near the arrows. Which are event types that trigger the state change.

The most important states are described below this paragraph.

3.1.1 Init State

The init state takes care of setting userBalance to 0, configures the GUI window and component triggers. Normally the E_PLAY event will be caused when the machine has initialized successfully. For this project, the system assumes the initialization always goes perfectly. And the user needs to create the E_PLAY event himself with a button.

3.1.2 Stakeamount state

This is the most important state because the main functions are started from within this state. The user should start with adding a balance first before he can do anything. If the user has balance, a gamble can be triggered or withdraw balance.

The program will be built with protection that disables the possibility of triggering events, which require balance.

3.1.3 Gamble state

The most extensive state is the gamble state. There are many different components involved: gamble machine, balance checking and logging of transaction.

3.1.4 Remaining states

The remaining states have a concrete goal and are self described.

3.2 Architecture Layer

This paragraph gives an insight into the architecture side of the system. The architecture layer visualizes the relationships between different subsystems. In Figure 3 is the architecture layer diagram visible and for more detail the most important parts of the architecture are explained.

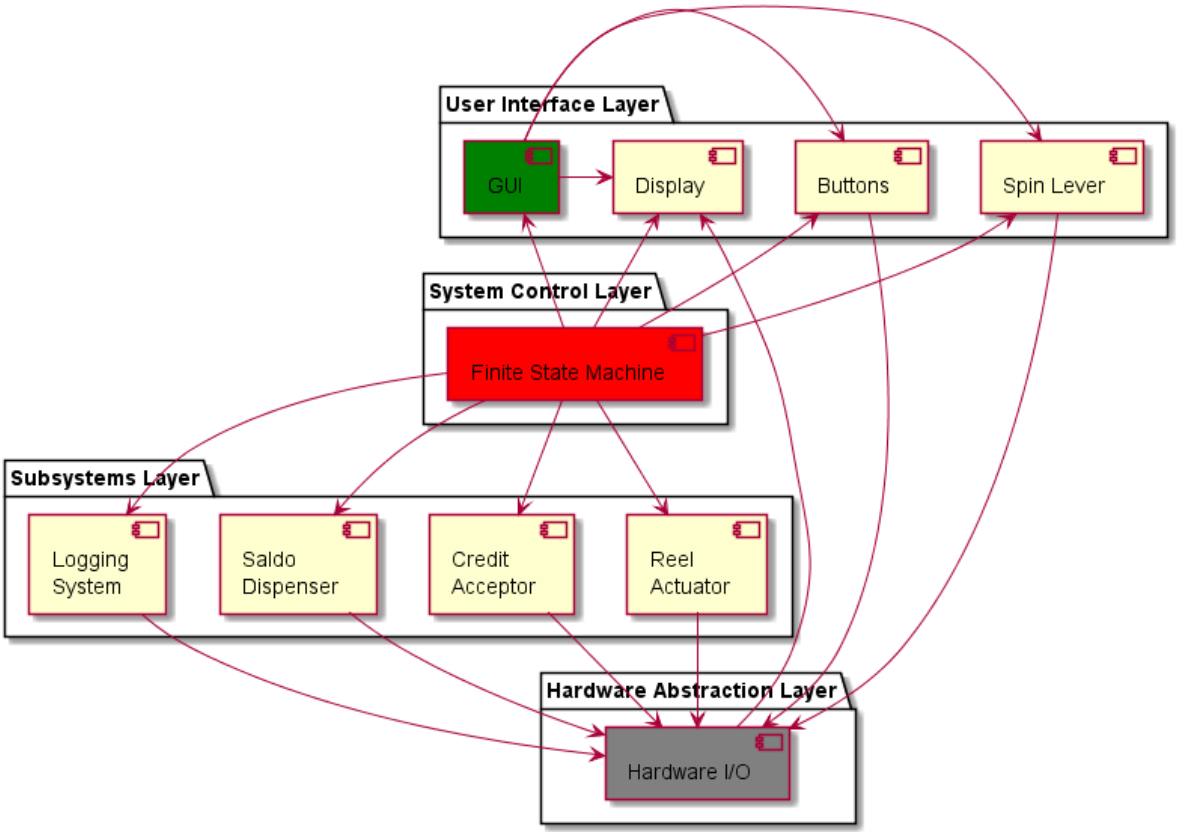

Figure 3 architecture layer diagram

3.2.1 User Interface Layer

This layer gives the user the possibility to communicate and interact with the system. The userinterface layer acts as a view and controller which the user interacts with.

3.2.2 System Control Layer

The slot machine is build according to a finite state machine principle, the management of states and events is done by this layer. The system control layer manages the events and determines what should happen.

3.2.3 Hardware Abstraction Layer

The hardware abstraction layer (HAL) simulates the hardware I/O. The scope within this project is to control the system without the hardware interactions. So, the GUI application simulates the hardware components and does react as a hardware component.

3.3 User Interface

Sketch of the physical case and GUI layout.

3.3.1 Physical Product

Figure 4 productcase concept

To keep in mind the interface of the product, a concept of the slot machine product has been made. Which is visible in Figure 4.

This sketch helps the development to keep a vision towards the product.

3.3.2 GUI HAL Concept

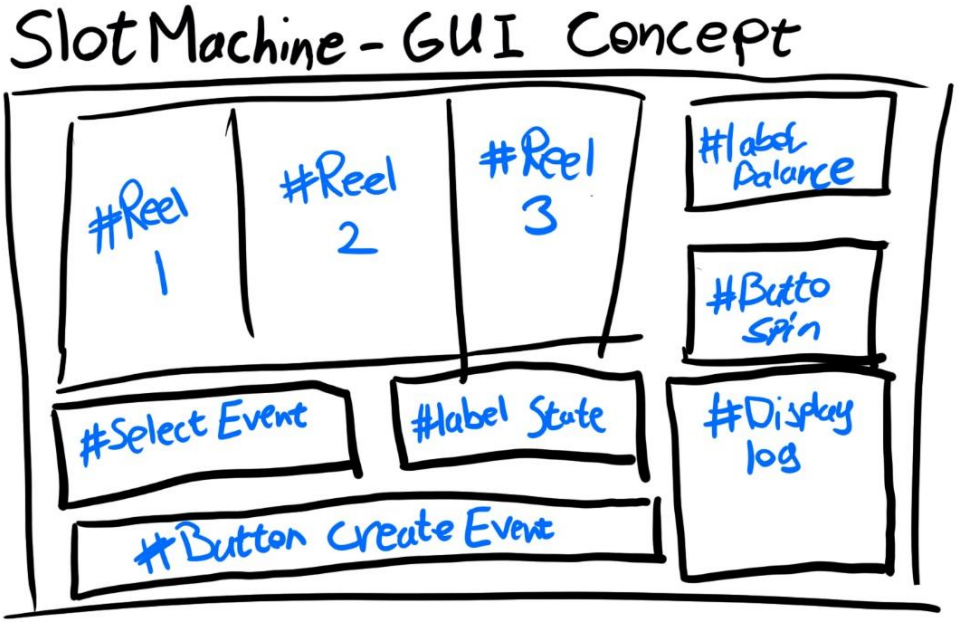

Figure 5 slot machine GUI layout concept sketch

The reel section has 3 reel index numbers. Together the reels decide the payout, later in section 4.3 game mechanics the working is described.

To generate events the user (integrate engineer) has the option to select an event that's available within the current state. To generate or trigger the event he should click the create event button.

3.4 System Limits

The development within this project has the following limits:

3.4.1 System Properties

Title	Description	Parameters
Pseudo Random	The system generates pseudo random generation only, which is acceptable for the scope of this project.	*Generator seed*: **time(null)** *Values between*: **0 ≤ x < 5**
Reels	The system has 3 fixed reels with symbols/characters on it.	*Reel amount:* **3**
Unique Reels	Each reel is **not** identical in index length and/or symbols index.	*Unique symbols reel:* **4** *Reel indexes length:* **3-7**
Statistical Model	The statistical model will be reused from Associate Professor Bradley Sward.[Ref 1]	*Re-use of existing model*
Profit margin	To give the host an indication and insight into the revenue possibilities the system assures a guaranteed profit margin.	*7% profit.*

Table 4 system limits table overview

3.4.2 System Features

Title	Description	Parameters
Stake money	The system gives the user the option to add two different fixed amounts at a time.	Adding choice: €50 or €100 selectable
Gamble	The system can start a gamble, when the user wants to bet his stake.	Caused by a Spin event

Return on bet	The system will determine the payout factor based on reels combinations	0 < X < 300% on the user bet amount

3.5 Business Logic & Game concept

The business model of this slot machine is based on a probability calculation of chances. For this project the decision has been made to re-use a simple model, from Professor Bradley Sward (**Ref 1**).

Bradley Sward has explained in a video how this probability model works, and he shared an excellent Excel template. The shared template is attached to this document as an appendix.

3.5.1 Basic Game Principle

The model starts off with identifying every possible combination. The number of combinations determines based on the amount of reel index places. A symbol index is an unique identifier which can reference to a specific picture or illustration on a reel.

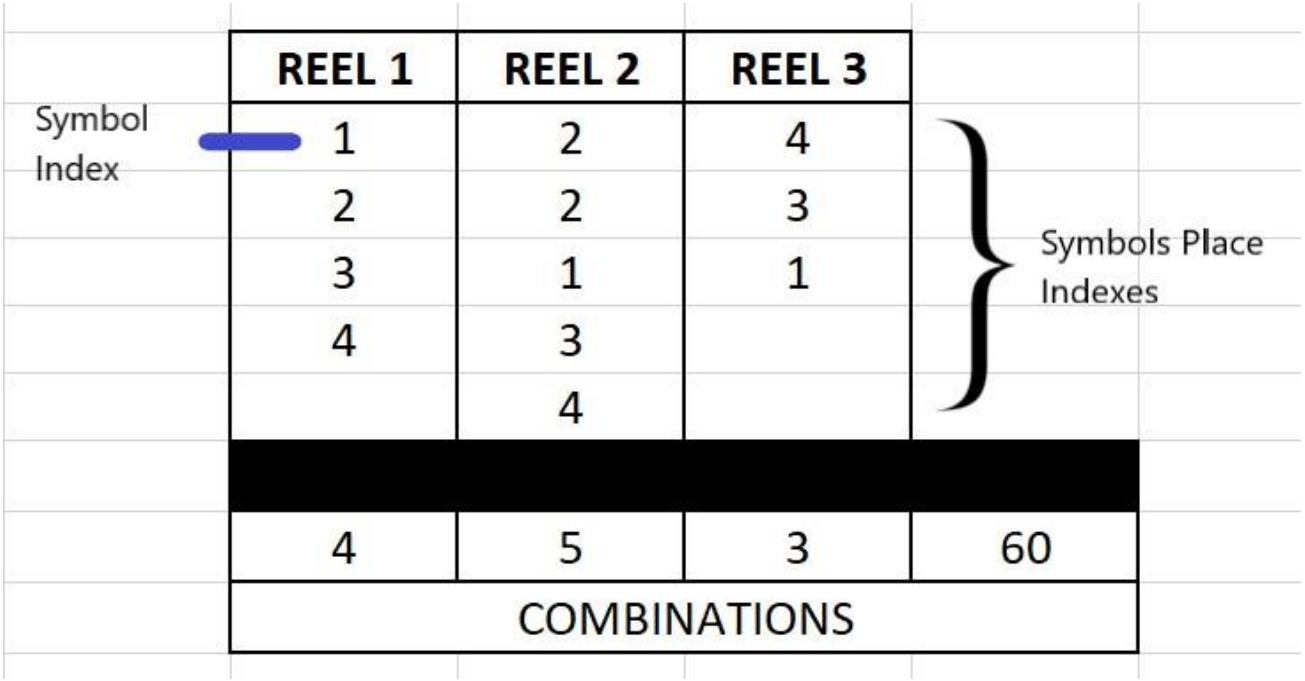

Figure 6 reels indexes overview

In Figure 6 reels indexes overview is visible that reel one is classified as expected: first index has first unique icon and numbering goes ordered. However, reel two is classified differently, it has five index places, but the system has only four unique symbols. So, there should be a double symbol, and indeed the unique symbol index two is at first and second index place.

There are 60 unique possible combinations, it's the product of reel index places.

Next step is to define the payout combinations. In Figure 7 you can find the payout combinations.

11	PAYS	1
22	PAYS	1
111	PAYS	3
222	PAYS	X
333	PAYS	15
444	PAYS	30

Figure 7 payout combinations

The payouts are not randomly defined, rather they are carefully configured so the machine host/owner is guaranteed to make money in the long term. In Figure 8 is the calculation for the average revenue viewable.

TOTAL PAYOUTS		56
TOTAL COMBINATIONS		60
PAYOUT PERCENTAGE		93%

Figure 8 statistical medium profit calculation

The total payout is the sum of payout factors from each combination. So, for every 11X the total payout increases with one, and so on.

What the calculation means for the user and host is: the user knows (or should know) that the longer he plays the more likely he is going to only get back 93% of the stake amount he puts into the game. The host knows that the more games there are played the more likely he gets an average return of 7% out of the machine.

3.6 Acceptance Testing

The developed system needs to be verified and validated, before the delivery for the client. This test is called acceptance test and is composed in co-operation with the client (normally).

The test strategy will be white box testing. Because the developer (student) is performing the tests himself. Some subprograms are tested in unit testing.

The definition of the acceptance test can be found below, Table 5 acceptance testreport definition.

Project: MoneyHeistMachine		**Code version:** vX.Y	**Date:** MM-DD-20YY	**ID:** #ZZ
Tester: name		**Notes**:		
ID	**Acceptance Requirement**	**Test-case**	**Parameters required**	**Hypothesis**
F1.1	States and events are logged.	System test	States are correct with a timestamp.	During GUI operation the states display is viewable by test engineer. When program is done and closed the file: adminLogs.txt has documented.
F2.1	Money is added with fixed money amounts.	System test	+ 50 EU + 100 EU	Balance + 50 EU Balance + 100 EU
F2.2	The user can generate a spin event, if he has balance.	System test	Without balance not allowed to start gamble	
F2.5	Money can be withdrawn if there is a balance	System test	Current balance will be paid out.	
F2.6	Balance is visible and up to date in every state.	System test	Balance display changes if balance itself changes.	
F2.7	Reels combination is clearly visible	System test	Triggered after E_SPIN event.	There will be first randomized effects and thereafter static end combination.
F2.8	The recovery option is defined as: the user can refund his money without trouble.	System test		User will be prevented from creating events that should not happen.
F3.1	The reels reproduce different combination results.	Unit test or System test	10 x spins and compare if there is a reel match.	The reels could have one exact same result, but not likely.
F3.3	Payout calculation is valid according to the business model.	Unit test or System test	Payout is applied as described	Balance changes after a gamble based on payout factor.
T1	System does not crash	System test	System prevents itself from crashing.	System does not crash unexpected.

Table 5 acceptance testreport definition

4. Realisation phase

This chapter gives an inside into the realization of the design plans and clients requirements. In paragraph 4.1 is GUI interface described. If you are interested in the program explanation, check out paragraph 4.2. Lastly the simple algorithm of gamble mechanism is documented at 4.3.

4.1 Graphical User Interface

The GUI is buildup in three different parts, which are explained below Figure 9.

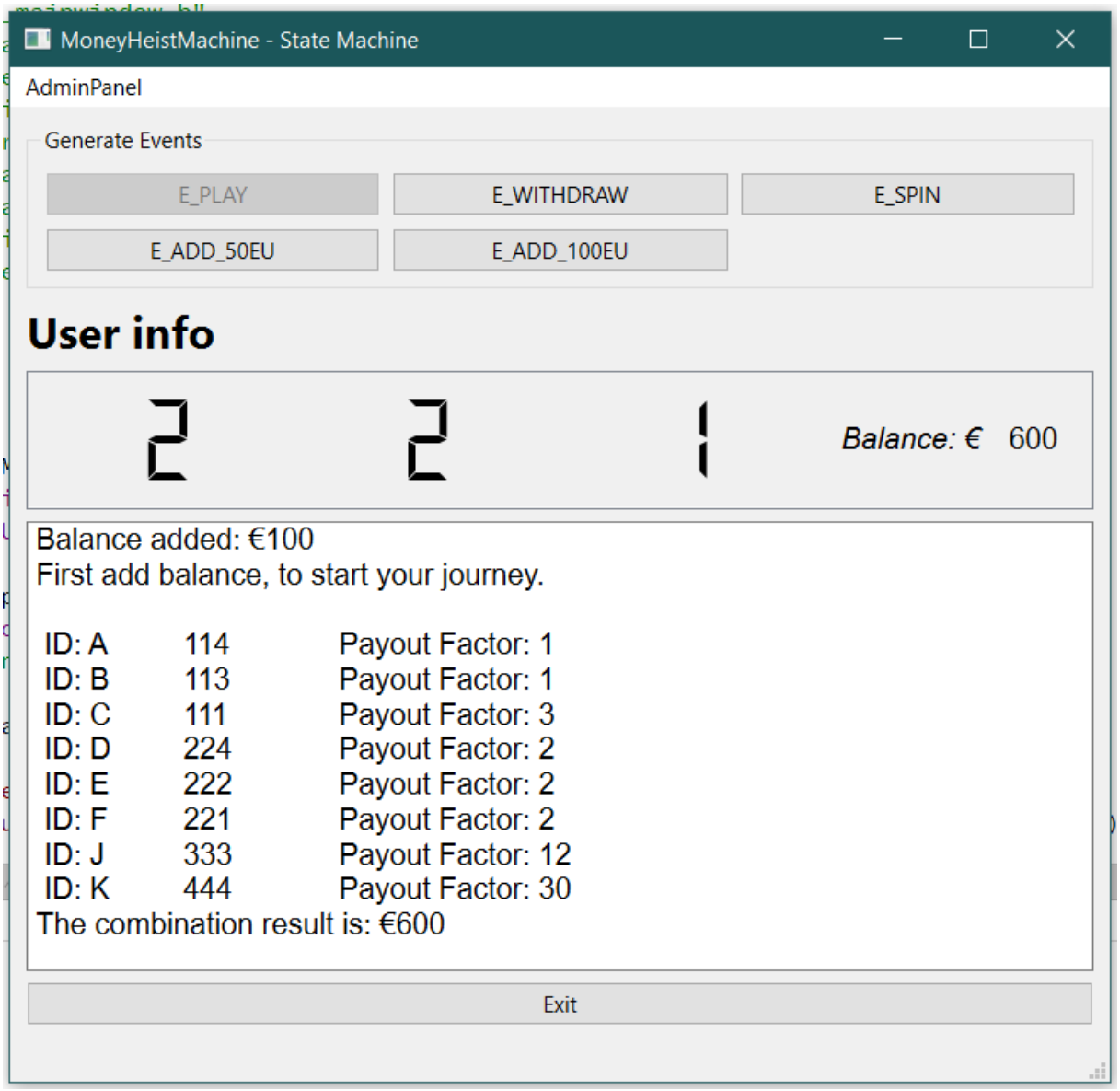

Figure 9 MoneyHeistMachine GUI v2.1

4.1.1 Generate Events Section

In this GUI section/group the user can manually generate hardware events which can cause a state change. To provide the user convenience and improve system stability, the system disables some of the events buttons. This prevents the user to generate an event that should not be able.

4.1.2 User Information Section

This layer handles viewing the user the essential information. The user should only see meaningful information to him, and the technical side is abstracted to him. This section tries to provide the user a user-friendliness environment by simple copywriting without technical jargon.

4.1.3 Admin Mode Section

When the system does not operate correctly, a system engineer should check what the system is doing. The GUI provides the service engineer with special admin tools, like a state logger. Which can be found in Figure 10.

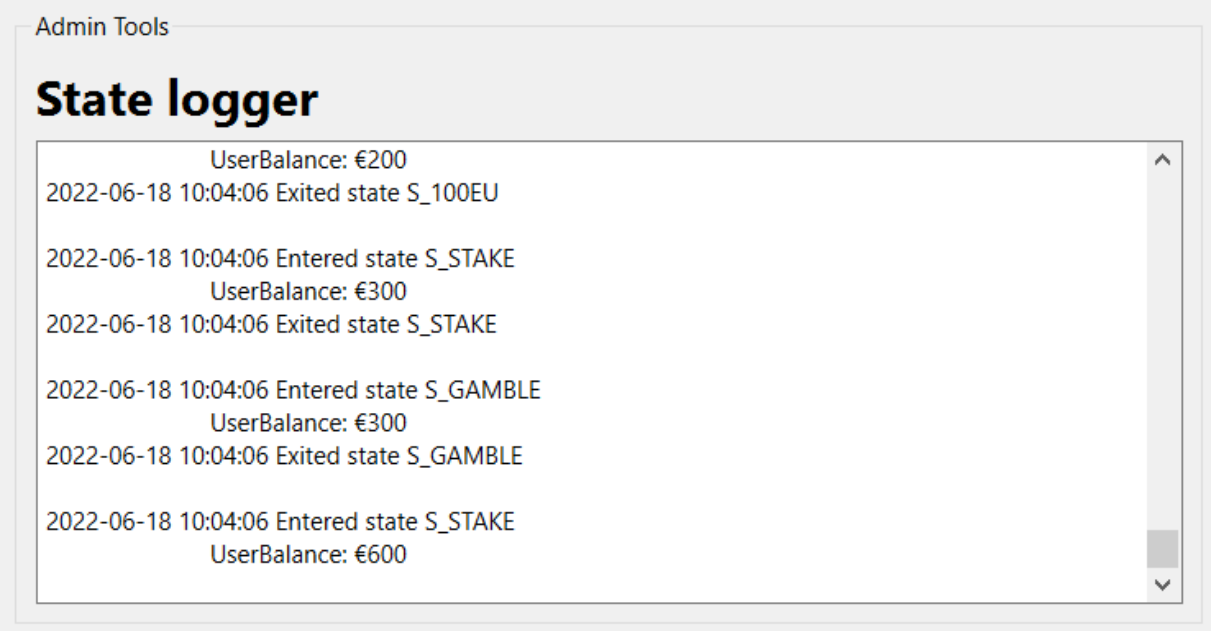

Figure 10 admin tools

In user manual you can find the instructions to enable admin tools.

4.2 Program Walkthrough

This chapter discusses the program in short. This is done with help of examples.

4.2.1 MainWindow Class

The MainWindow class is the most important piece in the program of the MoneyHeistMachine. Every state and user interface functionality are created within this.

To provide an example how this class is implemented check Figure 11 example MainWindow function: S_STAKE_enteredexample function and the description below it.

```
void MainWindow::S_STAKE_entered(void)
{
    /// Code Logic and actions for the state:
    ui->userBalance->setText(QString::number(userBalance));
    ui->E_ADD_100EU->setEnabled(true);
    ui->E_ADD_50EU->setEnabled(true);

    if(checkBalance()) {
        ui->E_SPIN->setEnabled(true);
        ui->E_WITHDRAW->setEnabled(true);
    }

    /// Admin logging state information process
    ui->userInfo->appendPlainText("First add balance, to start your
journey.\n");

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss
");
    logstring += "Entered state S_STAKE";
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);
}
```

Figure 11 example MainWindow function: S_STAKE_entered

The MainWindow class is build mostly with state functions. Every state has a on entry and exit function. On entry the function first updates the userBalance label, so the user gets can see his current balance. Secondly the user interface is updated, event buttons that are possible within this state are enabled and others disabled.

Furthermore, the user gets a message in the display with a subtle instruction to what to do next.

Lastly the function makes a state log for admin display.

4.2.2 GambleEngine

The GambleEngine code has three functions. The most important function is given in Figure 12.

```
int gambleResult(int betAmount) {
    int* reelsCombination = gambleCombination();
    int factor = determinePayout(reelsCombination,sizeof(reelsCombination));
    betAmount *= factor;
    return betAmount;
}
```

Figure 12 gambleResult function

GambleCombintation() gives an array of three random numbers. Each array number stands for a reel with unique symbol identifier. To determine the payout factor the function determinePayout() checks whether there is a matching combination, if so the corresponding payout factor is applied.

4.3 Game Mechanics

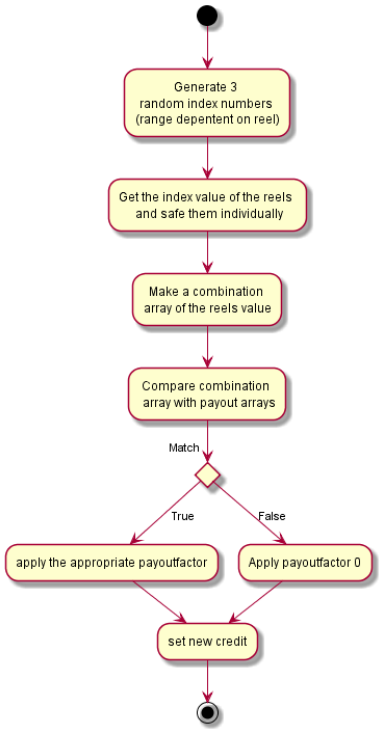

Figure 13 flowchart Gamble generation

Figure 13 shows the flowchart of the algorithm behind the reel generator. The reels themselves are constant arrays that can be changed if necessary. This also applies to the payout combinations and the payout factors.

5. Testing and Verifications

In this chapter the acceptance test report is documented. The results are discussed at paragraph 6.1.

Project: MoneyHeistMachine	**Code version:** v2.1	**Date:** 6-18-2022	**ID:** #01
Tester: Richard Kroesen	**Notes**:		

ID	Requirement title	Input	Output	Result
F1.1	AdminLog	Generation of every event and state transition.	adminLogs.txt	PASS
F2.1	Balance Insert	E__ADD_50EU E_ADD_100EU	Balance + 50 Balance + 100	PASS
F2.2	Gamble Event	E_SPIN button click event.	• Reel combination • Balance change • Back to S_STAKE state	PASS
F2.5	Balance Withdraw	E_WITHDRAW	Balance to zero. Withdrew amount is the balance before event.	PASS
F2.6	Balance visibility	Events dependent on Balance	Balance changes as expected	PASS
F2.7	Reels visibility	E_SPIN button click event.	Reels should give the first 3 sec random numbers as a reel effect. After that they should match the debugger output value	PASS
F2.8	User recovery protection	E_WITHDRAW after E_ADD_XXEU	User gets a full refund	PASS
F3.1	Reels randomization	E_SPIN button click event.	Reels should give the first 3 sec random numbers.	PASS
F3.3	Payout validation	E_STAKE entered after E_PLAY button click event for example.		PASS
T1	System stability	No crashes during testing every state and events possibilities	No crashes.	PASS

Table 6 acceptance test execution results

6. Conclusions

6.1 Results

As seen in the acceptance test report the system passed every test case. The system has been developed according to the requirements.

If you want to use the program yourself, please take a look at the user manual: Appendix 2 User manual.

6.2 System Expansion Vision

Title	Description	Improvment result
Customized Icons	Instead of viewing index combination numbers, pleasant illustration could be used.	More user engagement & excitement.
User and admin loging	To log more specific details and protect the system for counterfeits a proper login form is preferred.	Protection and machine Administration
Variable money insertion		More option and convenience for the user: easy access to mobile or card payments
Gamification	• Visual / Graphical Effects. • Audio elements. • Leaderboards. • Skill based modes within certain reels combination.	More engagement and interaction

Appendix 1 Program listings

MainWindow

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStateMachine>
#include <QDateTime>
#include <QDebug>
#include <QHistoryState>
#include <QPropertyAnimation>
#include <QParallelAnimationGroup>
#include <QRandomGenerator>
#include <QFile>
#include <QMessageBox>
```

Figure 14 MainWindow class includes (mainwindow.h)

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    bool checkBalance(void);
    void updateBalance(int amount);
    void setBalance(long amount);

    // Machine States
    QState *S_INIT;
    QState *S_STAKE;
    QState *S_PAYOUT;
    QState *S_GAMBLE;
    QState *S_50EU;
    QState *S_100EU;

    // State transition functions
private slots:
    void S_INIT_entered(void);
    void S_INIT_exited(void);

    void S_STAKE_entered(void);
    void S_STAKE_exited(void);

    void S_PAYOUT_entered(void);
    void S_PAYOUT_exited(void);

    void S_GAMBLE_entered(void);
    void S_GAMBLE_exited(void);

    void S_50EU_entered(void);
    void S_50EU_exited(void);
    void S_100EU_entered(void);
    void S_100EU_exited(void);

    void adminLogCheck(void);

private:
    Ui::MainWindow *ui;

    // Global private variable, volatile so the compiler is optimiziting
carefully
    volatile long userBalance;

    // Initilization of statemachine function
    void createStateMachine(void);

    // function to write logfiles
    void fileLogger(const QString &text);

    QStateMachine slotStateMachine;
};
```

Figure 15 mainwindow class declaration (mainwindow.h)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

extern "C" {#include "GambleEngine.h"}

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    ,ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    MainWindow::setWindowTitle("MoneyHeistMachine - State Machine");
    //    MainWindow::setWindowIcon();

    createStateMachine();

    ui->AdminPanel->setDisabled(true);
    connect(ui->actionShow_State_Logger, SIGNAL(triggered()), this,
SLOT(adminLogCheck()));
}

MainWindow::~MainWindow()
{
    fileLogger(ui->StateLog_display->toPlainText());
    qDebug() << "GUI MainWindow closed";
    delete ui;
}

bool MainWindow::checkBalance(){return userBalance > 0 ? true : false; }

void MainWindow::updateBalance(int amount){ userBalance += amount; }

void MainWindow::setBalance(long amount){ userBalance = amount; }

void MainWindow::S_INIT_entered(void)
{
    /// Code Logic and actions for the state:
    setBalance(0);

    /// Admin logging state information process.
    QString logstring;
    QString userName = "Test Engineer";

    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Entered state: S_INIT\n";
    logstring += "\tUser: ";
    logstring += userName;
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);

    /// User interface update process
    ui->userInfo->setPlainText("Welcome, please click \"E_PLAY\" to start gambling
mode.");
}

void MainWindow::S_INIT_exited(void)
{
    /// Admin logging state information process.
    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_INIT\n";
    ui->StateLog_display->appendPlainText(logstring);

    /// User interface update process
    if(userBalance > 0) {
        QMessageBox::warning(this, "System Fault","Error: The balance is not
zero.");
        qDebug() << "System error: balance not init as zero.";
    }
    ui->E_PLAY->setDisabled(true);
}
```

```
void MainWindow::S_INIT_exited(void)
{
    /// Admin logging state information process.
    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_INIT\n";
    ui->StateLog_display->appendPlainText(logstring);

    /// User interface update process
    if(userBalance > 0) {
        QMessageBox::warning(this, "System Fault","Error: The balance is not
zero.");
        qDebug() << "System error: balance not init as zero.";
    }

    ui->E_PLAY->setDisabled(true);
}

void MainWindow::S_STAKE_entered(void)
{
    /// Code Logic and actions for the state:
    ui->userBalance->setText(QString::number(userBalance));
    ui->E_ADD_100EU->setEnabled(true);
    ui->E_ADD_50EU->setEnabled(true);

    if(checkBalance()) {
        ui->E_SPIN->setEnabled(true);
        ui->E_WITHDRAW->setEnabled(true);
    }

    /// Admin logging state information process
    ui->userInfo->appendPlainText("First add balance, to start your journey.\n");

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Entered state S_STAKE";
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);
}

void MainWindow::S_STAKE_exited(void)
{
    char payOutTableString[600];

    for(int i = 0; i < PAYOUT_TYPES_AMOUNT; i++) {
        sprintf(payOutTableString + strlen(payOutTableString),"\n ID: %c\t",
payOutsConstants[i].letterId);

        for(int j = 0; j < REELS_AMOUNT; j++) {
            sprintf(payOutTableString+ strlen(payOutTableString),"%d",
payOutsConstants[i].arrayCombination[j]);
        }
        sprintf(payOutTableString+ strlen(payOutTableString),"\t Payout Factor: %d",
payOutsConstants[i].payOutFactor);
    }

    /// User interface update process
    ui->userInfo->setPlainText("\t= PAYOUT TABLE = \n" +
QString::fromStdString(payOutTableString) + "\n");
    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_STAKE\n";
    ui->StateLog_display->appendPlainText(logstring);
}
```

```
void MainWindow::S_PAYOUT_entered(void)
{
    ui->E_ADD_100EU->setDisabled(true);
    ui->E_ADD_50EU->setDisabled(true);
    ui->E_SPIN->setDisabled(true);
    ui->E_WITHDRAW->setDisabled(true);

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Entered state S_PAYOUT";
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);

    int moneyOut = userBalance;
    userBalance = 0;
    ui->userInfo->setPlainText("You withdrew: €" + QString::number(moneyOut));

    ui->userInfo->appendPlainText("You saved your money. Wise choice.\n");
    ui->userInfo->appendPlainText("You are going back to stake state, \nso if you
want to take your changes again, go for it!\n");
}

void MainWindow::S_PAYOUT_exited(void)
{
    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_PAYOUT\n";
    ui->StateLog_display->appendPlainText(logstring);
}
void MainWindow::S_GAMBLE_entered()
{
    ui->userInfo->clear();
    ui->userInfo->setPlainText("$$$ GAMBLE STARTED $$$");

    QTime dieTime= QTime::currentTime().addSecs(1);
    while (QTime::currentTime() < dieTime)
        QCoreApplication::processEvents(QEventLoop::AllEvents, 100);

    ui->userInfo->clear();

    QParallelAnimationGroup *reelGroupAnimation = new QParallelAnimationGroup;
    QPropertyAnimation *reelOneAnimation = new QPropertyAnimation(ui->ReelOne,
"value");
    QPropertyAnimation *reelTwoAnimation = new QPropertyAnimation(ui->ReelTwo,
"value");
    QPropertyAnimation *reelThreeAnimation = new QPropertyAnimation(ui->ReelThree,
"value");

    reelGroupAnimation->addAnimation(reelOneAnimation);
    reelGroupAnimation->addAnimation(reelTwoAnimation);
    reelGroupAnimation->addAnimation(reelThreeAnimation);

    reelOneAnimation->setDuration(600);
    reelTwoAnimation->setDuration(610);
    reelThreeAnimation->setDuration(597);

    reelOneAnimation->setStartValue(QRandomGenerator::global()->bounded(10));
    reelTwoAnimation->setStartValue(QRandomGenerator::global()->bounded(10));
    reelOneAnimation->setStartValue(QRandomGenerator::global()->bounded(10));

    QString logstring;
    long userBetAmount = 0;
```

```
    if(checkBalance()) {

        logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
        logstring += "Entered state S_GAMBLE";
        logstring += "\n\tUserBalance: €";
        logstring += QString::number(userBalance);

        userBetAmount = userBalance;
        updateBalance(-userBetAmount);

        int* comboArray = gambleCombination();

        qDebug() << comboArray[0] << comboArray[1] << comboArray[2];
        logstring += "\tReels: " + QString::number(comboArray[0]) +
QString::number(comboArray[1]) + QString::number(comboArray[2]);

        reelOneAnimation->setEndValue(comboArray[0]);
        reelTwoAnimation->setEndValue(comboArray[1]);
        reelThreeAnimation->setEndValue(comboArray[2]);
        reelGroupAnimation->start();

        int factor = determinePayout(comboArray, sizeof(comboArray));
        int gambleResult = userBetAmount * factor;
        qDebug() << factor << " " << gambleResult;
        updateBalance(gambleResult);

        logstring += "\tPayout Factor: " + QString::number(factor);
        ui->userInfo->appendPlainText("Your Payout Factor is: ");

        QTime dieTime= QTime::currentTime().addSecs(1);
        while (QTime::currentTime() < dieTime)
            QCoreApplication::processEvents(QEventLoop::AllEvents, 100);

        ui->userInfo->appendPlainText("\t" + QString::number(factor) + "X");

        logstring += "\tGamble Outcome: €" + QString::number(gambleResult -
userBetAmount) + "\n";
        ui->userInfo->appendPlainText("Gamble Outcome: €" +
QString::number(gambleResult - userBetAmount) + "\n");

        ui->StateLog_display->appendPlainText(logstring);
    } else {
        logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
        logstring += "Err. E_NO_BALANCE.";
        ui->StateLog_display->appendPlainText(logstring);
    }
}

void MainWindow::S_GAMBLE_exited()
{
    ui->E_SPIN->setDisabled(true);
    ui->E_WITHDRAW->setDisabled(true);

    ui->userInfo->appendPlainText("Thanks for Playing.\n");

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_GAMBLE\n";
    ui->StateLog_display->appendPlainText(logstring);
}
```

```
void MainWindow::S_50EU_entered()
{
    ui->userInfo->appendPlainText("Balance added: €50");

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Entered state S_50EU";
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);
}

void MainWindow::S_50EU_exited()
{
    updateBalance(50);

    QString logstring;

    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_50EU\n";
    ui->StateLog_display->appendPlainText(logstring);
}

void MainWindow::S_100EU_entered()
{
    ui->userInfo->appendPlainText("Balance added: €100");

    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Entered state S_100EU";
    logstring += "\n\tUserBalance: €";
    logstring += QString::number(userBalance);
    ui->StateLog_display->appendPlainText(logstring);
}

void MainWindow::S_100EU_exited()
{
    updateBalance(100);
    QString logstring;
    logstring = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ");
    logstring += "Exited state S_100EU\n";
    ui->StateLog_display->appendPlainText(logstring);
}

void MainWindow::adminLogCheck()
{
    if(ui->AdminPanel->isEnabled()) {
        ui->AdminPanel->setDisabled(true);
        qDebug() << "Admin Tools disabled";
    } else {
        ui->AdminPanel->setDisabled(false);
        qDebug() << "Admin Tools Enabled";
    }
}
```

```
void MainWindow::createStateMachine()
{
    // Create States.
    S_INIT = new QState();
    S_STAKE = new QState();
    S_PAYOUT = new QState();
    S_GAMBLE = new QState();
    S_50EU = new QState();
    S_100EU = new QState();

    // Add Events/transistions.
    S_INIT->addTransition(ui->E_PLAY, &QPushButton::clicked, S_STAKE);

    S_STAKE->addTransition(ui->E_WITHDRAW, &QPushButton::clicked, S_PAYOUT);
    S_STAKE->addTransition(ui->E_SPIN, &QPushButton::clicked, S_GAMBLE);
    S_STAKE->addTransition(ui->E_ADD_50EU, &QPushButton::clicked, S_50EU);
    S_STAKE->addTransition(ui->E_ADD_100EU, &QPushButton::clicked, S_100EU);

    S_PAYOUT->addTransition(S_STAKE);
    S_100EU->addTransition(S_STAKE);
    S_50EU->addTransition(S_STAKE);
    S_GAMBLE->addTransition(S_STAKE);

    // Add states to the machine.
    slotStateMachine.addState(S_INIT);
    slotStateMachine.addState(S_STAKE);
    slotStateMachine.addState(S_PAYOUT);
    slotStateMachine.addState(S_GAMBLE);
    slotStateMachine.addState(S_50EU);
    slotStateMachine.addState(S_100EU);

    slotStateMachine.setInitialState(S_INIT);

    // connect transitions with functions.
    connect(S_INIT, &QState::entered, this, &MainWindow::S_INIT_entered);
    connect(S_INIT, &QState::exited, this, &MainWindow::S_INIT_exited);

    connect(S_STAKE, &QState::entered, this, &MainWindow::S_STAKE_entered);
    connect(S_STAKE, &QState::exited, this, &MainWindow::S_STAKE_exited);

    connect(S_GAMBLE, &QState::entered, this, &MainWindow::S_GAMBLE_entered);
    connect(S_GAMBLE, &QState::exited, this, &MainWindow::S_GAMBLE_exited);

    connect(S_PAYOUT, &QState::entered, this, &MainWindow::S_PAYOUT_entered);
    connect(S_PAYOUT, &QState::exited, this, &MainWindow::S_PAYOUT_exited);

    connect(S_100EU, &QState::entered, this, &MainWindow::S_100EU_entered);
    connect(S_100EU, &QState::exited, this, &MainWindow::S_100EU_exited);
    connect(S_50EU, &QState::entered, this, &MainWindow::S_50EU_entered);
    connect(S_50EU, &QState::exited, this, &MainWindow::S_50EU_exited);

    slotStateMachine.start();

    qDebug() << "GUI MainWindow started";
    qDebug() << "Slot Machine started";
}

void MainWindow::fileLogger(const QString &text) {
    QFile logFile("adminLogs.txt");

    if(!logFile.open(QFile::WriteOnly | QFile::Text)) {
        QMessageBox::warning(this,"title","file not open");
    }
    QTextStream log_out(&logFile);
    log_out << text << "\n";
    logFile.close();
}
```

GambleEngine

Written in C, because it is originally programmed for PROG2 class.

```c
#ifndef GAMBLEENGINE_H
#define GAMBLEENGINE_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

// Constants to prevent magic numbers:
#define REELS_AMOUNT            3
#define REEL_ONE_INDEXES        4
#define REEL_TWO_INDEXES        5
#define REEL_THREE_INDEXES      3
#define PAYOUT_TYPES_AMOUNT     8

// Reels index configuration constants:
const int reelOne[REEL_ONE_INDEXES] = {1, 2, 3, 4};
const int reelTwo[REEL_TWO_INDEXES] = {2, 2, 1, 3, 4};
const int reelThree[REEL_THREE_INDEXES] = {4, 3, 1};


// Struct for gamble payout combination:
typedef struct PayoutCombination {
    const char letterId;
    const int arrayCombination[REELS_AMOUNT];
    const int payOutFactor;
} PayoutCombination_t;

extern PayoutCombination_t payOutsConstants[PAYOUT_TYPES_AMOUNT];

/// Function Prototypes

// Generate Random number
int generateIndexNumber(int limit);

// Get the payout factor dependent on the reels combination
int determinePayout(int combo[], int length);

// Returns a array of the reels gamble combination
int* gambleCombination();

// calculates the gamble credit result
int gambleResult(int betAmount);

const char* payOutTable();

#endif // GAMBLEENGINE_H
```

```
#include "GambleEngine.h"

// Payout combinations constants:
PayoutCombination_t payOutsConstants[PAYOUT_TYPES_AMOUNT] = {
    {'A', {1,1,4}, 1},
    {'B', {1,1,3}, 1},
    {'C', {1,1,1}, 3},
    {'D', {2,2,4}, 2},
    {'E', {2,2,2}, 2},
    {'F', {2,2,1}, 2},
    {'J', {3,3,3}, 12},
    {'K', {4,4,4}, 30}
};

int gambleResult(int betAmount) {
    int* reelsCombination = gambleCombination();
    int factor = determinePayout(reelsCombination,
sizeof(reelsCombination));
    betAmount *= factor;
    return betAmount;
}

int* gambleCombination() {
    static int reelsArray[REELS_AMOUNT]; // static so the function can
return the int array

    // Random index numbers generated within the range of the reels
    int randomnumber = generateIndexNumber(REEL_ONE_INDEXES);
    int randomnumber2 = generateIndexNumber(REEL_TWO_INDEXES);
    int randomnumber3 = generateIndexNumber(REEL_THREE_INDEXES);

    // Index of the gamble combination assignment from random reels indexes
    reelsArray[0] = reelOne[randomnumber];
    reelsArray[1] = reelTwo[randomnumber2];
    reelsArray[2] = reelThree[randomnumber3];

    return reelsArray;
}

int generateIndexNumber(int limit) {
    int randomIndex = 0;
    srand(time(NULL));
    randomIndex = (rand() % limit);
    return randomIndex;
}

int determinePayout(int combo[], int length) {
    int payoutFactor = 0;

    for(int i = 0; i < PAYOUT_TYPES_AMOUNT; i++) {
        if(memcmp(combo, payOutsConstants[i].arrayCombination, length+1) ==
0) {
            payoutFactor = payOutsConstants[i].payOutFactor;
        }
    }
    return payoutFactor;
}
```

Appendix 2 User manual

Program version: v3.0

User:

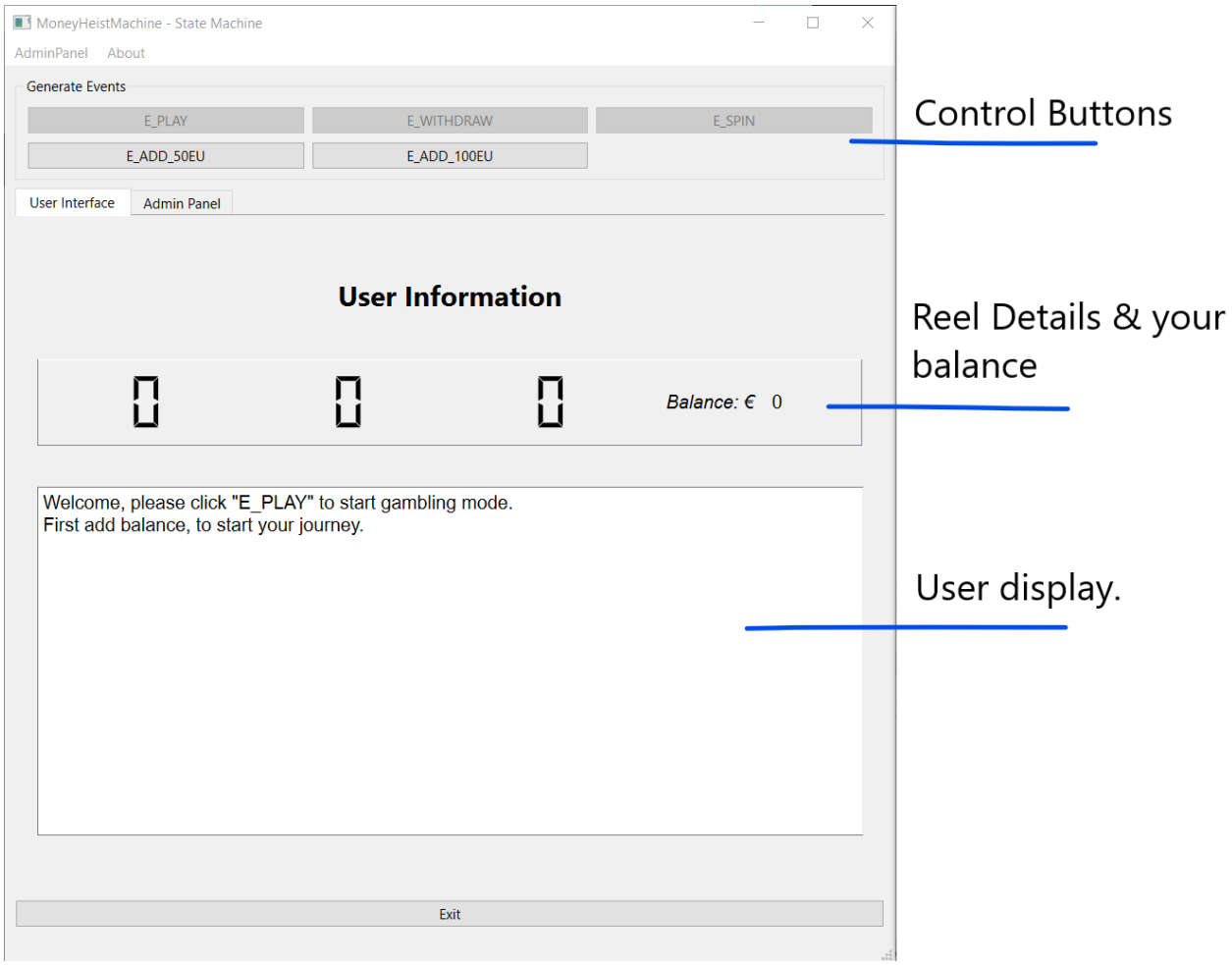

Figure 16 userinterface layout

When the program is started, follow these steps

1. Click “E_PLAY” *control button* to start the slot machine interface.
2. Follow the instructions given on the *User Display*.
3. To exit the system savely, please click the “Exit” button or the cross above.
   WARNING! If you close the program your balance is lost. If this happens by accident please consult a technical service engineer or admin of the machine.

Admin

- To enable admin tools, go to AdminPanel menu (as in Figure 17) and click on “StateLogger” label.
- To disable you follow the same instruction, the tools will be toggled when you click on the label again.
- To check if the tools are enabled check if the label has a checkmark.

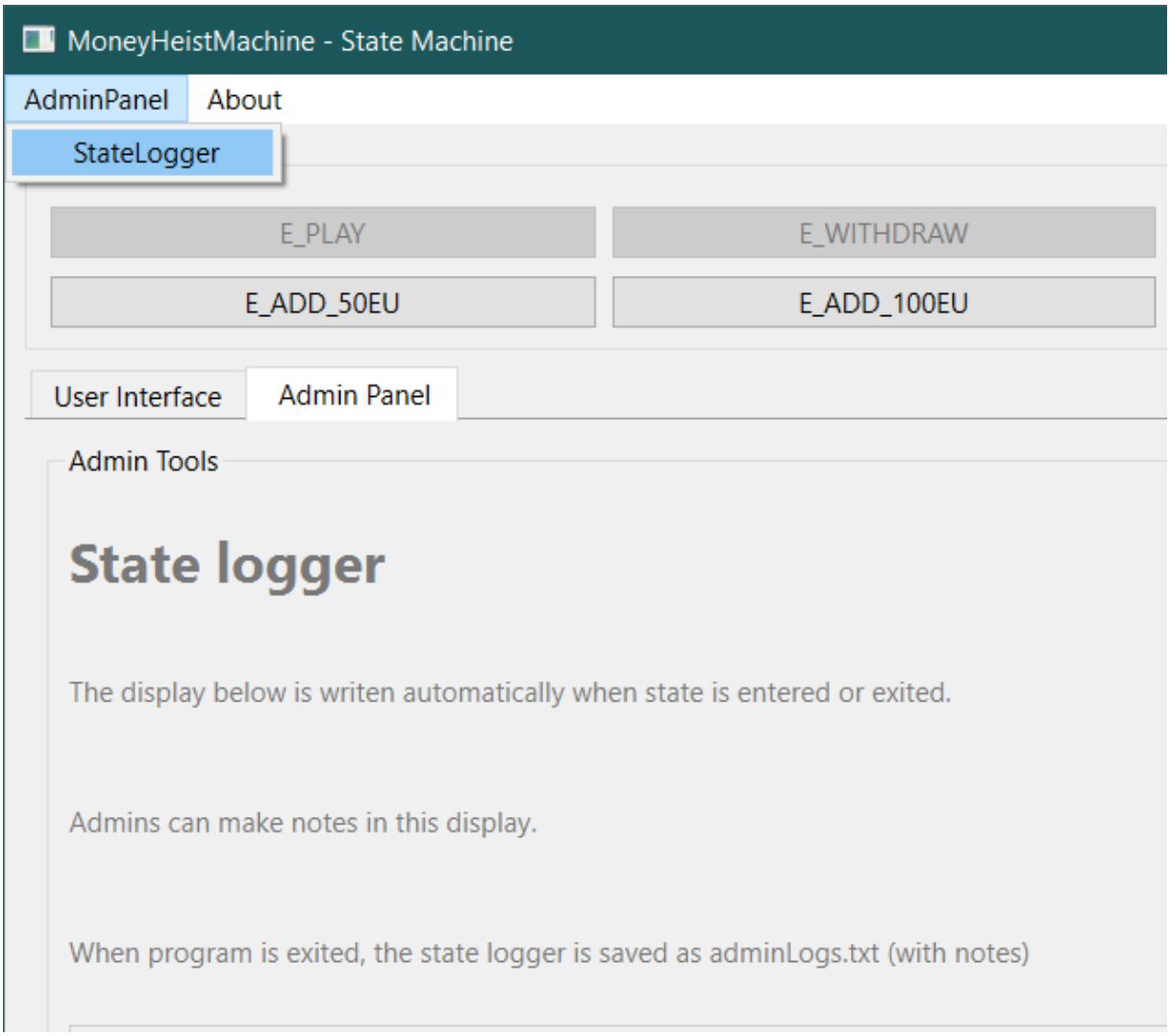

Figure 17 admin tools enabler

References

Without the shared knowledge on the internet this project would not be as it is now. So many thanks to the creators.

The references are made in the APA 7e edition format. The references are clickable to find the place of use.

Ref 1: Slot machine math model.

Bradley, B. [Bradley Sward]. (2021, 5 februari). The Basic Mathematics of Slot Machines [Video]. YouTube. https://www.youtube.com/watch?v=JyIWQIdxaOA