Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

Requirements analysis

1. Introduction
The story of Steve is a story-driven psychological horror game that is as close to reality as possible, putting the player in the role of Steve - a man in the midst of a personal crisis. Accompanied through the day and his nightmarish experiences, the player delves deep into Steve's reality and his fragile psyche.
The working name of the game is “**GetOut**”
The official name is “**Kenopsia**”

The points that haven’t been mentioned yet can’t be filled out at this given point of time in the development process.

1.1. Purpose of the system
Creating an immersive, terrifying experience for the player. The system encompasses various components, each serving a specific purpose to enhance the horror elements.
This game is based on the principle of a walking simulator where the main purpose is to explore the world by simply walking around.

1.2. Objectives and success criteria of the project
Create a game using an engine and documenting the process of development.
In the end all participants should be able to understand the steps of game-specific software development processes and be able to organize themselves as a team.

1.3.1 Definitions, acronyms, and abbreviations
Steve: the working name of the main character who is played in first person
Boss: Steve’s boss who will occur in the fifth scene of the game, at work. He will be a side character.
Wife/Horror lady: divorced wife of Steve, she will only occur in the fourth scene. She is the main antagonist and will be included in the main part of the story.
Smiler: a grinning face which looks very scary that will occur as an emoji in the fourth scene at home. This is a part of the scary illusions that the player will face during the game.

Game Objects: The following list contains all items that are defined as objects in the game.
- 2 Bottles laying on the couch
- Letter laying on the table
- Pan in the kitchen
- Chocolate Chunks Package in kitchen

- Plate in the kitchen
- Wardrobe in living room
- Front door
- Desk in office
- Computer on desk
- Smiler
- Security cameras
- Horror lady
- Hitbox in scene 4 main entrance and kitchen
- bed scene 4

Definition of Hitboxes:
Throughout the game there are hitboxes where when the player enters, an event will occur such as starting an audio file or moving another object.

1.3.2 Technical definitions

Feedback message:
Should be a small text at the top left of the screen. The messages ensure an unproblematic gameplay and give advice for the player to have a better user experience.

StateManager:
The class that defines the different stages and saving points throughout the game.

Interactable:
The interface that defines the function for objects to be interacted with. It provides features like distance for interaction and a “checkup” method to prove if an object is in range.

PlayerInteractionComponent:
The class that handles the interaction of the player with other objects having an Interactable-Interface.

Defined interaction range:
public float interactRange = 5f is the actual range at which the player is able to interact with the predefined object. The range is based on the player position.

Player Movement:
The class that implements the ability for the player to move by using the keyword.

UI Manager:

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

The class that updates the Graphic Interface by analyzing the data from the StateManager.

Animation Manager:
The class for all general animations throughout the game.

Menu:
Composition of classes that create the main menu in the beginning when starting the game.

1.4. References
Trello
GitHub
StoryBoard
Usability (see appendix)

2. Proposed system
2.1. Overview
- Game should be able to run on Windows/MAC/Linux
- All developers are using Windows
- Game will be programmed and designed with Unity

Proposed input devices:
- Mouse
- Keyboard

Proposed mouse and Keyboard layout:
The game should be programmed for the standard German QWERTZ keyboard layout.
The game should be programmed for a typical computer mouse only taking left click, right click and movement of the mouse into consideration.

Proposed keys in usage:
WASD, R, E, Esc, left mouse button, Shift

2.2. Functional requirements

General gameplay:
- Player wakes up on the couch, then he has to accomplish assignments eg. tidying up the kitchen
- The player has to dress up in the bedroom so that he can go to work
- At work he sits on his desk and falls asleep

- Inside of his dream the set is way darker and the player should get the feeling of being watched
- While exploring the house the player notices strange anomalies and finally wakes up at work

ID: FR1	Title: Interact with surroundings
Description	User interacts with objects by using the left mouse button or “E” on the keyboard
Acceptance Criterion	- User triggers event with mouse or keyboard clicking followed by a specified action
Notes	Will be called in update function throughout the game (List of objects can be found in **1.4 Definitions**) (Type of keyboard layout and mouse can be found in **2.1 Overview**)

ID: FR2	Title: Move in World
Description	User can move by using W,A,S,D
Acceptance Criterion	- User must be able to move in all directions except for jumping - Movement is developed for FP (First person) - Movement speed is constant and can change between walking and running speed using shift
Notes	Player can only move within the defined areas of the scenes (Type of keyboard layout can be found in **2.1 Overview**)

ID: FR3	Title: Enter Gameplay
Description	User must be able to load gameplay
Acceptance Criterion	- User must be able to start // TODODODODOD or resume gameplay
Notes	Requires functional memory management

//TODO Andrei probiert aus ob man das noch gescheit programmieren kann

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

ID: FR4	Title: SaveGameManager
Description	User is able to save at any given point throughout the game but only the beginning of the current scene is being saved. When escaping to menu the current scene is saved as well as when changing to a new scene.
Acceptance Criterion	- After saving the game, there is a file containing the current state of the story.
Notes	Handle storageManager with enumerations for simple access.

2.3 Nonfunctional Requirements

ID: NFR1	Title: Performance
Description	Make sure that game runs smoothly at all times
Acceptance Criterion	1. The game should have a smooth gameplay, even during complex scenes and effects, therefore the game should have different graphic settings (low, medium, high)
Notes	For each graphic setting, we apply shadow and light in different resolutions.

ID: NFR2	Title: Load time
Description	Make sure that user doesn't have to wait too long to load into the game
Acceptance Criterion	1. The game should load within 10 seconds when starting a new session
Notes	

ID: NFR3	Title: Usability
Description	The game should have understandable instructions in general

Acceptance Criterion	1. Every button should have a suitable description or name and suitable design 2. The user is getting introduced to the game without a direct tutorial (Indirect explanation) 3. At all time throughout the game the current assignment should be clear to the player
Notes	Will be tested with suitable Usability Tests. Usability description can be found in appendix.

ID: NFR4	Title: Reliability
Description	Title: The game must not crash or freeze during gameplay session
Acceptance Criterion	1. The game must not crash more than once per hour under normal conditions 2. The game must not crash more than once every half hours under stress conditions (e.g. maximum settings, switching between scenes) 3. The game must auto-save progress at regular intervals 4. Auto-save points should be created at key moments (Change of scenes)
Notes	All crashes and critical errors must be logged with detail information in order to debug

ID: NFR5	Title: Visual Quality
Description	The game should have a decent resolution
Acceptance Criterion	1. Game should work at low and high quality settings
Notes	

2.3.1 User Interface and human factors

In order to properly improve our user interface and user-friendliness we created a Usability-Test for scene 1 and 2, where we let other people play the game and ask whether there were difficulties or not.
In the following are the most important arguments of the test subjects:

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

Complaint: The text description in the upper corner didn’t stay long enough to read.
Solution: We increased the duration of appearance of the text description.

Complaint: It was not clear whether I did enough tasks to get out of the house or not.
Solution: We mentioned in the text description that it is now able to leave the house.

Complaint: I couldn’t see the key suggestion when I looked at an interactable object.
Solution: We made sure that every time the user looks at an interactable object a key suggestion appears in a size and color that can’t be overseen.

2.3.2 Documentation
Documentation is done using Doxygen

Can be found in the appendix of Report.

2.3.5 Error handling and extreme conditions

Changing of StoryBoard
Problem:
During the development process, we realized that the initial scope of the game was too ambitious. The original storyboard included too many features and elements, making it impossible to implement everything within our time and resource constraints.

Solution:
To stay on track, we made the decision to reduce the scope of the storyboard, focusing only on the core aspects that are essential to the game. This allowed us to streamline development and concentrate our efforts on completing the necessary requirements within the available timeline (timeline between meetings).

Starting too late with Doxygen Documentation
Problem:
After a few developing and implementation steps, it took us more and more time to relate the past ideas and methods we created. Whether we were modifying completed features or unfinished ones, we always had to invest a significant amount of time to fully understand and immerse ourselves in the code before making any changes.

Solution:

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

With the use of doxygen documentation we were able to improve our workflow and relate our written methods.

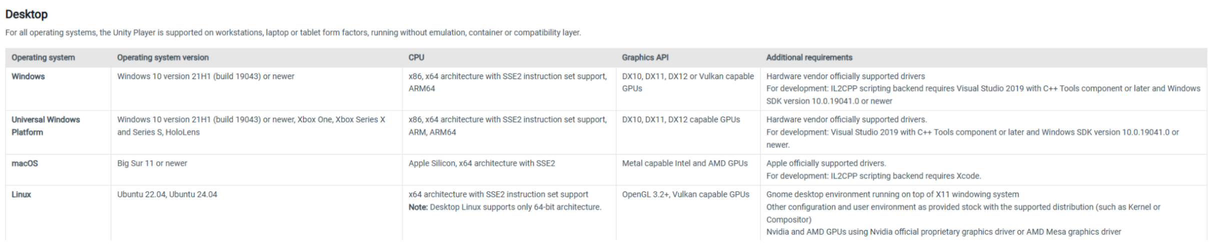
Wrong origin of ray-tracing
Problem:
Initially, the ray used for ray-intersection was originating from the center of the player model instead of the player's head, which caused difficulties in picking up objects and interacting with the environment since the angle between view and interaction ray was different.

Solution:
To fix this, we recalibrated the ray's origin point, moving it from the center of the model of the player's head. This adjustment greatly improved the precision of item interactions and overall gameplay experience.

2.3.7. System modifications

//TODO Andrei Änderungen des systems beschreiben (z.B. vom Prototypen zum clean code zu kommen)

2.3.8. Physical environment



Desktop

For all operating systems, the Unity Player is supported on workstations, laptop or tablet form factors, running without emulation, container or compatibility layer.

Operating system	Operating system version	CPU	Graphics API	Additional requirements
Windows	Windows 10 version 21H1 (build 19043) or newer	x86, x64 architecture with SSE2 instruction set support, ARM64	DX10, DX11, DX12 or Vulkan capable GPUs	Hardware vendor officially supported drivers For development: IL2CPP scripting backend requires Visual Studio 2019 with C++ Tools component or later and Windows SDK version 10.0.19041.0 or newer
Universal Windows Platform	Windows 10 version 21H1 (build 19043) or newer, Xbox One, Xbox Series X and Series S, HoloLens	x86, x64 architecture with SSE2 instruction set support, ARM, ARM64	DX10, DX11, DX12 capable GPUs	Hardware vendor officially supported drivers. For development: Visual Studio 2019 with C++ Tools component or later and Windows SDK version 10.0.19041.0 or newer.
macOS	Big Sur 11 or newer	Apple Silicon, x64 architecture with SSE2	Metal capable Intel and AMD GPUs	Apple officially supported drivers. For development: IL2CPP scripting backend requires Xcode.
Linux	Ubuntu 22.04, Ubuntu 24.04	x64 architecture with SSE2 instruction set support **Note:** Desktop Linux supports only 64-bit architecture.	OpenGL 3.2+, Vulkan capable GPUs	Gnome desktop environment running on top of X11 windowing system Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor) Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver

[Source]

2.3.10. Resource issues

Encryption Process (Andrei Costeniuc)

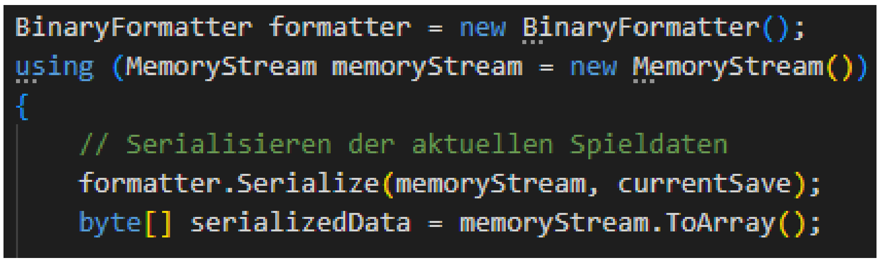
Overview

The save data is protected by encrypting it using the AES-256 encryption algorithm. This ensures that even if someone gains access to the save file, they cannot easily read or modify its contents without the encryption key. Additionally, a SHA.256 hash is used to detect tampering with the encrypted data.

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

Serializing the Save Data

Before encryption, the game's current state is serialized into a byte array using the BinaryFormatter. Serialization converts the SaveData object into a sequence of bytes that can be stored in a file:



```
BinaryFormatter formatter = new BinaryFormatter();
using (MemoryStream memoryStream = new MemoryStream())
{
    // Serialisieren der aktuellen Spieldaten
    formatter.Serialize(memoryStream, currentSave);
    byte[] serializedData = memoryStream.ToArray();
```

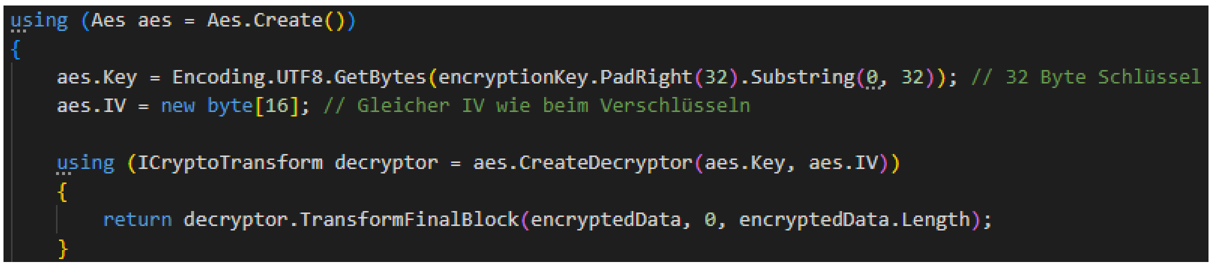
Encrypting the Serialized Data

The serialized byte array is encrypted using AES-256. AES (Advanced Encryption Standard) is a symmetric encryption algorithm that requires a secret key and an initialization vector to encrypt and decrypt data.

- The Key is a 32-character string, padded or trimmed to ensure the correct length.
- The Initialization Vector (IV) is a 16-byte array initialized to all zeroes. We initialized it to zero for simplicity, though a random IV can enhance security.

The Encrypt function takes the serialized data and:

1. Initializes the AES encryption engine
2. Creates an encryptor using the key and IV
3. Encrypts the byte array using TransformFinalBlock



```
using (Aes aes = Aes.Create())
{
    aes.Key = Encoding.UTF8.GetBytes(encryptionKey.PadRight(32).Substring(0, 32)); // 32 Byte Schlüssel
    aes.IV = new byte[16]; // Gleicher IV wie beim Verschlüsseln

    using (ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV))
    {
        return decryptor.TransformFinalBlock(encryptedData, 0, encryptedData.Length);
    }
```

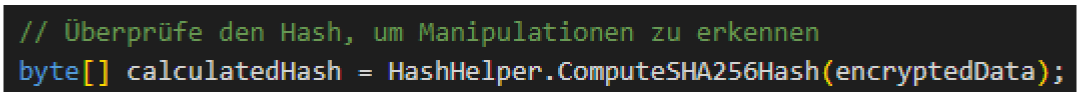
The result is a fully encrypted version of the serialized save data, which is unreadable without the correct key and IV.

Creating a Hash for Tamper Detection

To ensure that the save file has not been tampered with, a SHA-256 hash of the encrypted data is created. This hash acts as a unique "fingerprint" of the

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

encrypted data. When loading the save file, this hash is recalculated and compared with the stored hash to verify data integrity.

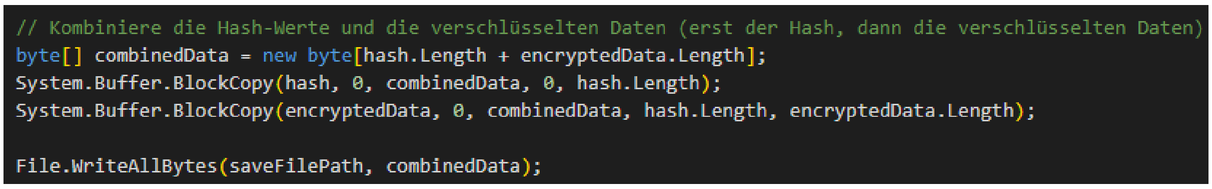


```
// Überprüfe den Hash, um Manipulationen zu erkennen
byte[] calculatedHash = HashHelper.ComputeSHA256Hash(encryptedData);
```

The ComputeSHA256Hash function creates a new SHA256 instance, computes the hash of the encrypted data and returns the 32-byte hash.

Combining the Hash and Encrypted Data

The hash and the encrypted data are combined into a single byte array - the first 32 bytes store the hash, the remaining bytes store the encrypted data. This combined array is then written to the save file.

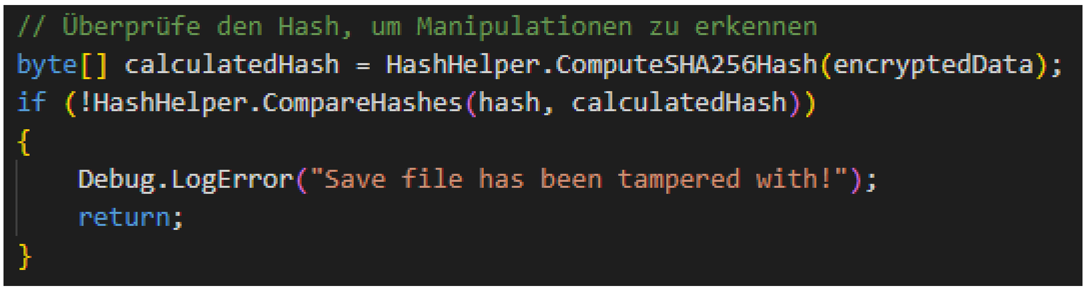


```
// Kombiniere die Hash-Werte und die verschlüsselten Daten (erst der Hash, dann die verschlüsselten Daten)
byte[] combinedData = new byte[hash.Length + encryptedData.Length];
System.Buffer.BlockCopy(hash, 0, combinedData, 0, hash.Length);
System.Buffer.BlockCopy(encryptedData, 0, combinedData, hash.Length, encryptedData.Length);

File.WriteAllBytes(saveFilePath, combinedData);
```

Save File Validation During Loading

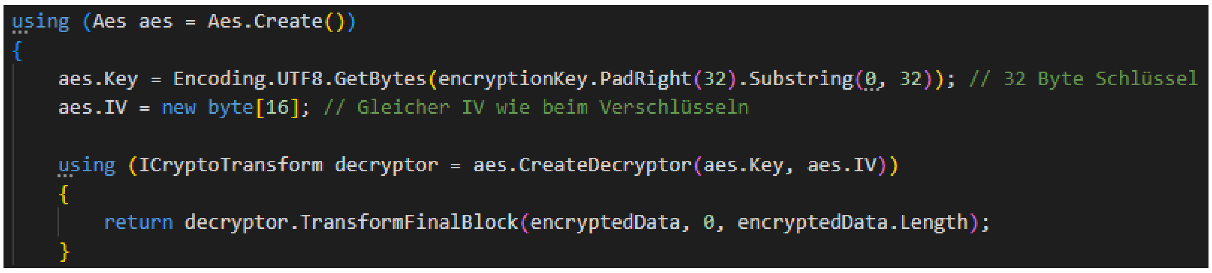
When loading the save file, the first 32 bytes are extracted as the hash, the remaining bytes are extracted as the encrypted data and a new hash is computed from the encrypted data and compared with the extracted hash. If the hashes do not match, it means the file has been tampered with and the load process is aborted.



```
// Überprüfe den Hash, um Manipulationen zu erkennen
byte[] calculatedHash = HashHelper.ComputeSHA256Hash(encryptedData);
if (!HashHelper.CompareHashes(hash, calculatedHash))
{
    Debug.LogError("Save file has been tampered with!");
    return;
}
```

Decrypting the Encrypted Data

If the hashes match, the encrypted data is decrypted using the same AES-256 algorithm with the same key and IV. The decrypted data is then deserialized back into a SaveData object.



```
using (Aes aes = Aes.Create())
{
    aes.Key = Encoding.UTF8.GetBytes(encryptionKey.PadRight(32).Substring(0, 32)); // 32 Byte Schlüssel
    aes.IV = new byte[16]; // Gleicher IV wie beim Verschlüsseln

    using (ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV))
    {
        return decryptor.TransformFinalBlock(encryptedData, 0, encryptedData.Length);
    }
```

2.4. Pseudo requirements

ID: FR_SCENE1	Title: Scene 1 “Home”
Description	The user has an environment simulating his home. He can freely walk through the house and interact with objects.
Acceptance Criterion	- Alpha Walls cover the complete house, so the player can't go outside - The player can interact with a letter, bottles and objects in the kitchen as well as the wardrobe and entrance door
Notes	

ID: FR_SCENE2	Title: Scene 2 “Office”
Description	An American style decorated office with views of other buildings in a city.
Acceptance Criterion	- Alpha Walls cover the complete house, so the player can't go outside - One computer is turned on and lightened up by external light sources.
Notes	

ID: FR_SCENE3	Title: Scene 3 “Office sitting”
Description	The user is at his desk (still in office) and can only rotate the camera.
Acceptance Criterion	- The player can look up and down with 80° and 90° - When interacting (clicking) with his desk, the player

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

	is sent to next scene
Notes	

ID: FR_SCENE4	Title: Scene 4 “Home - nightmare”
Description	The user has an environment simulating his home. He can freely walk through the house. It's dark and music is playing in background
Acceptance Criterion	- Music is playing in background environment is kept dark - Alpha Walls are placed, such that the player can’t exit the property of his house -
Notes	

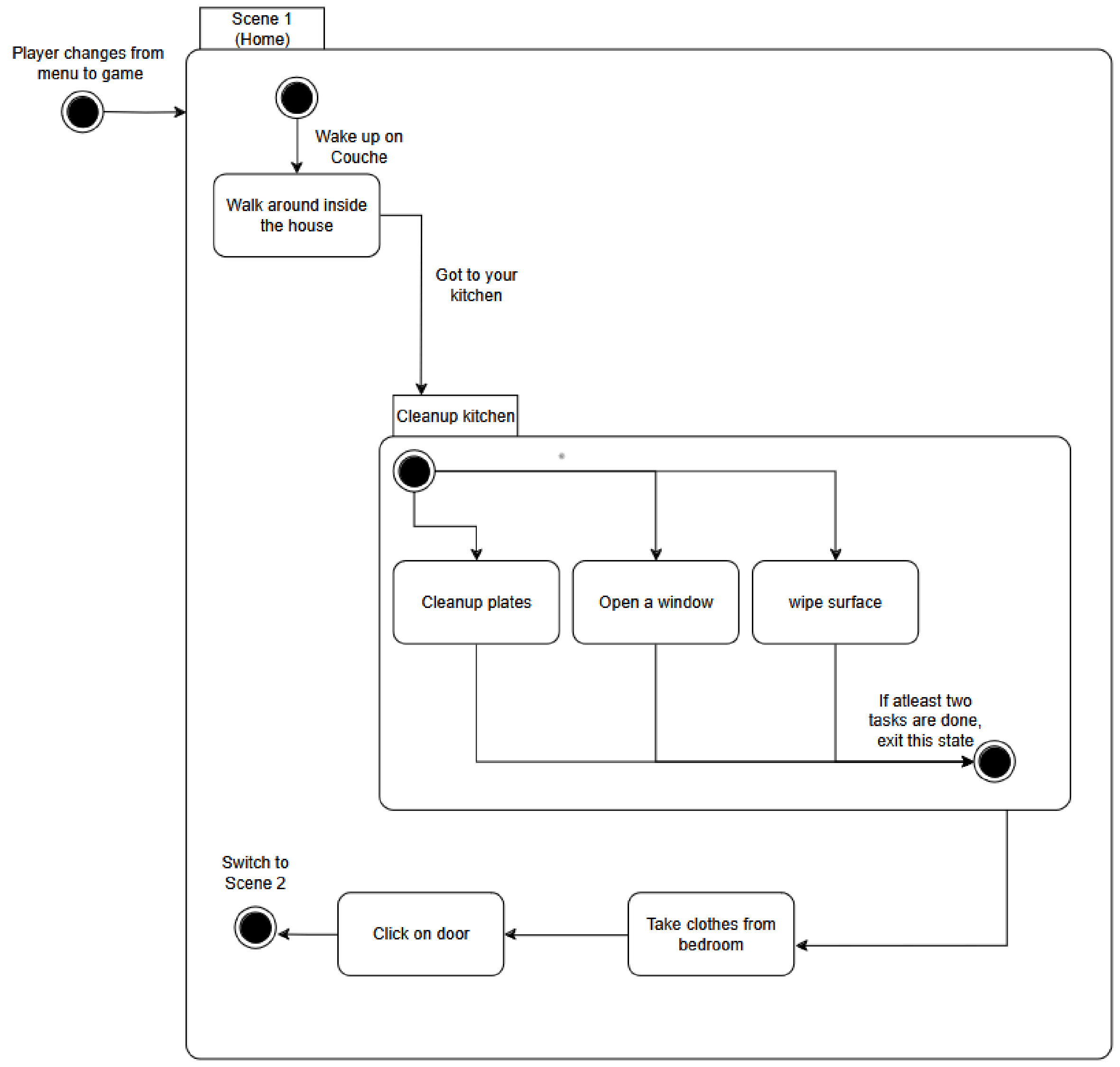
ID: FR_MENU	Title: Scene “Main Menu”
Description	The user can start the game, setup graphical settings and quit the game.
Acceptance Criterion	- Start button to change to scene 1. This one will always start the game from beginning. Meaning => Scene 1 - Settings button will change to graphical settings. The settings will be stored during the game runtime Quit button will stop the game
Notes	

2.5. System models

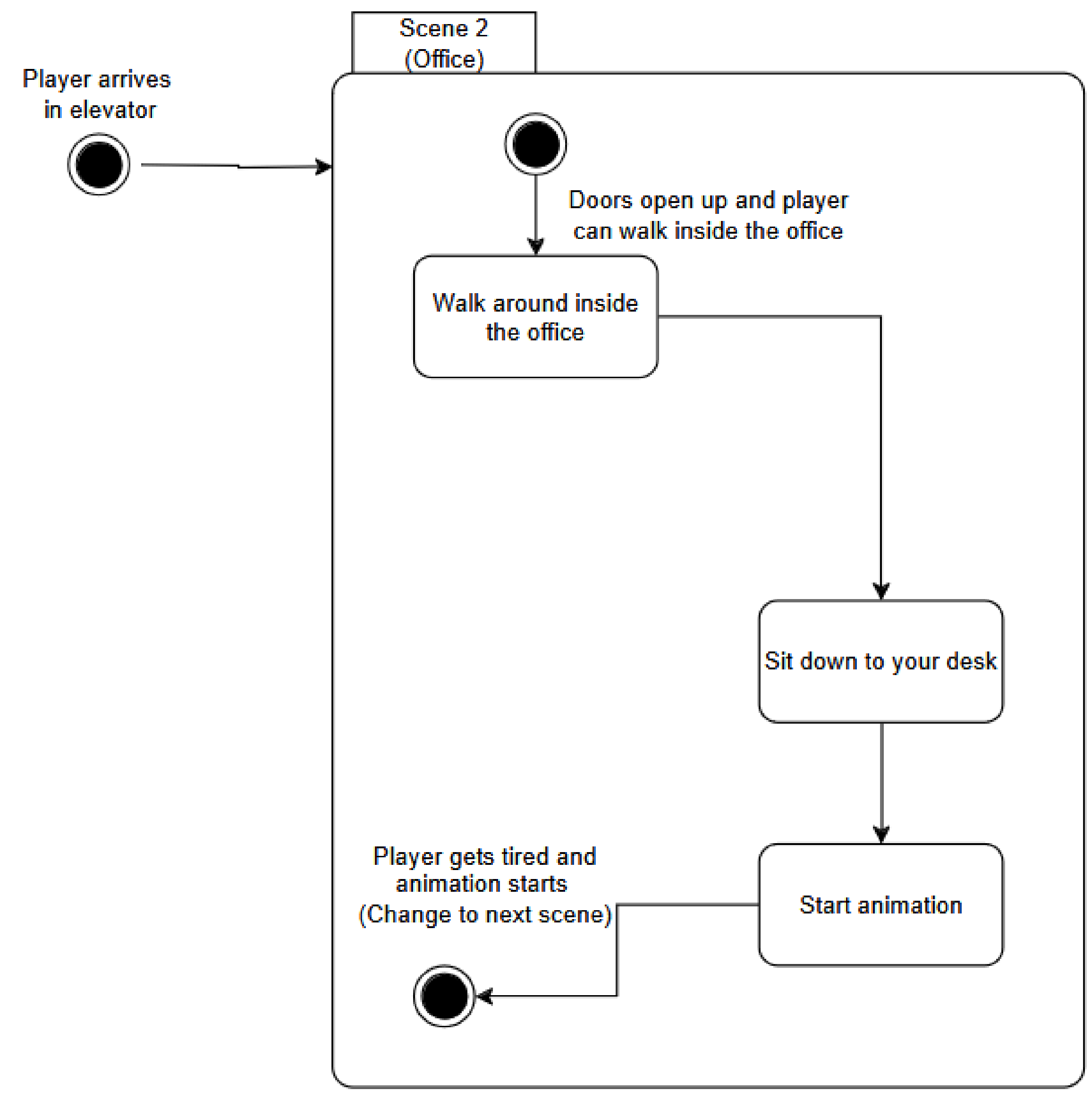
2.5.1 Scenarios

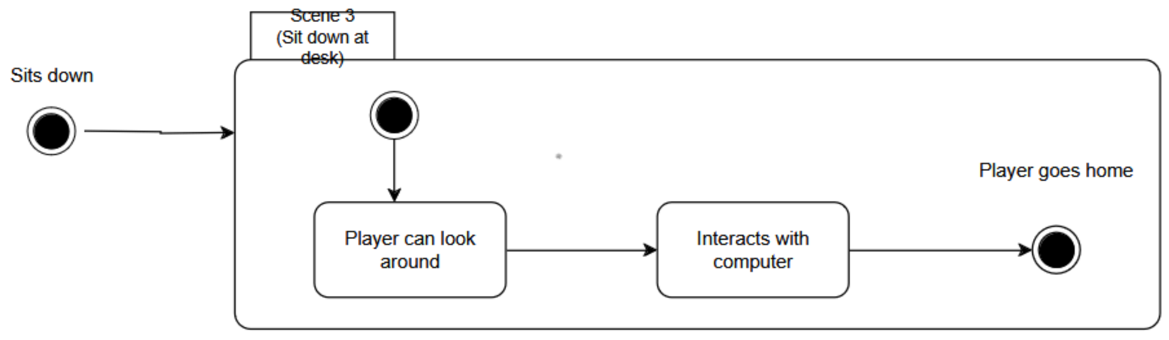
In order to easily describe the scenarios of the game we created StateMachineDiagrams that visually shows the possible interactions that the player can make and which interactions lead to the following scene.

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

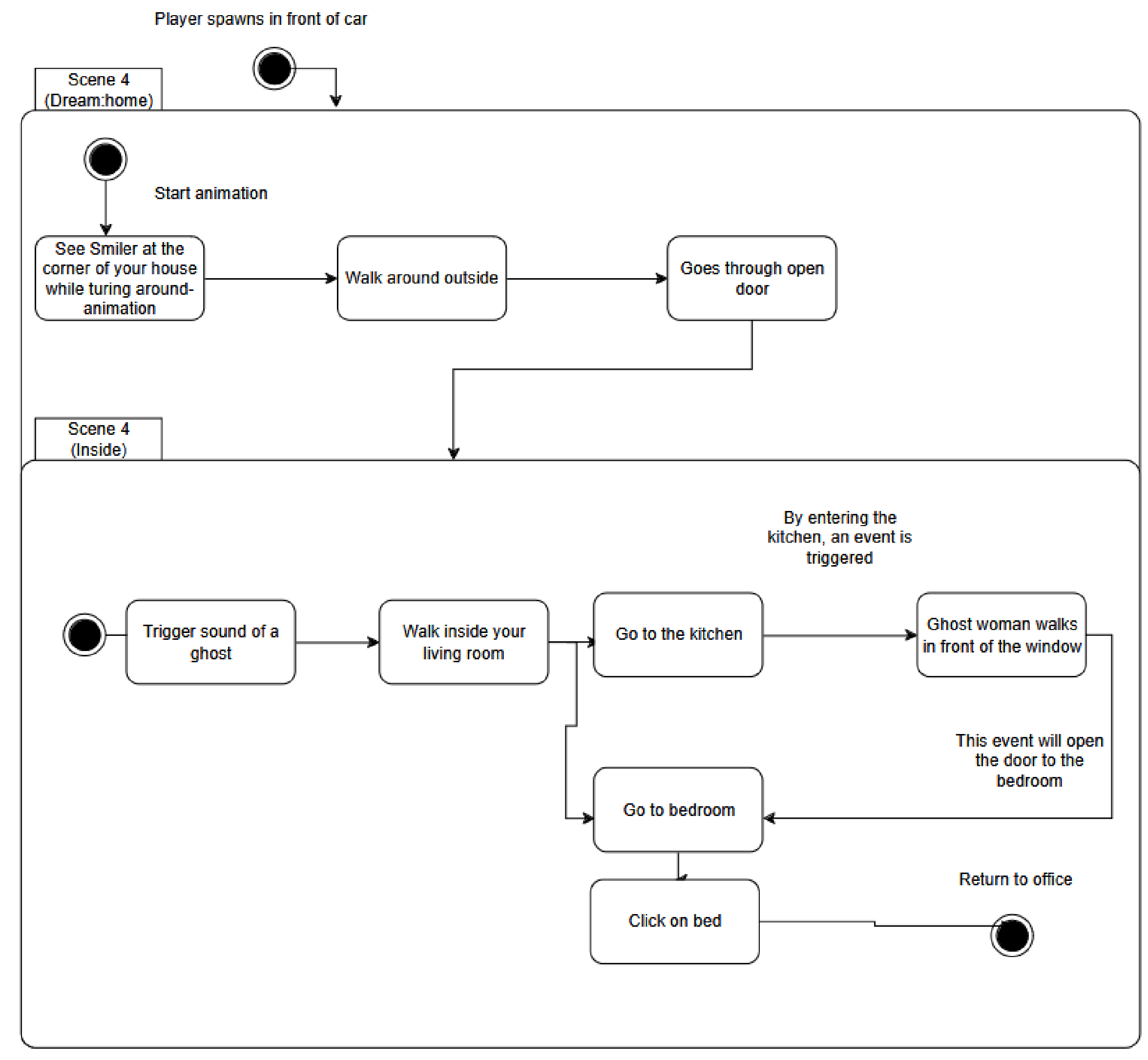


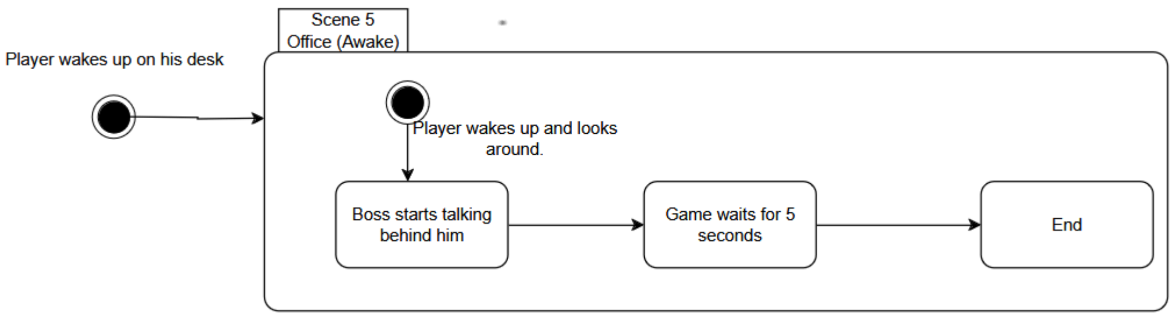
Christian Schäfer
Timo Skrobanek
Andrei Costeniuc


Scene 2 (Office)
Player arrives in elevator
Doors open up and player can walk inside the office
Walk around inside the office
Sit down to your desk
Start animation
Player gets tired and animation starts (Change to next scene)



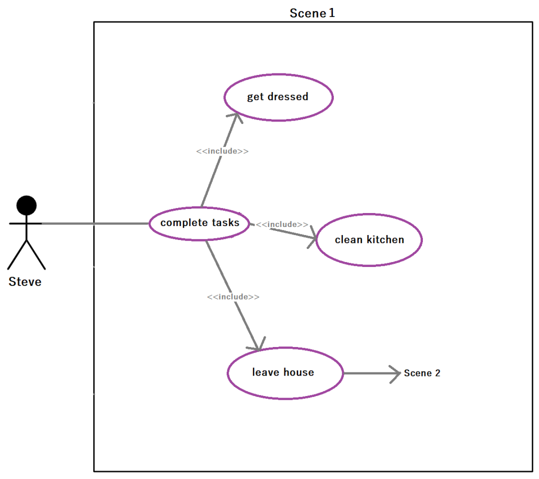
Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

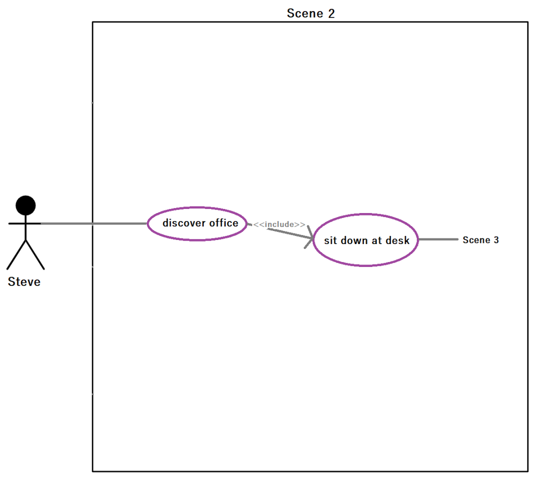

Player spawns in front of car
Scene 4 (Dream:home)
Start animation
See Smiler at the corner of your house while turing around-animation
Walk around outside
Goes through open door
Scene 4 (Inside)
By entering the kitchen, an event is triggered
Trigger sound of a ghost
Walk inside your living room
Go to the kitchen
Ghost woman walks in front of the window
This event will open the door to the bedroom
Go to bedroom
Click on bed
Return to office

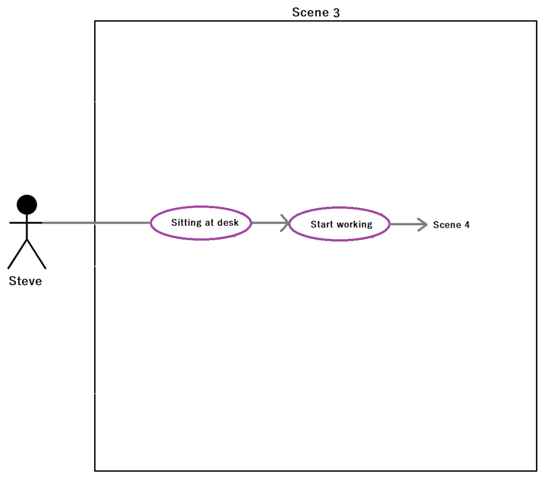

Scene 5 Office (Awake)
Player wakes up on his desk
Player wakes up and looks around.
Boss starts talking behind him
Game waits for 5 seconds
End

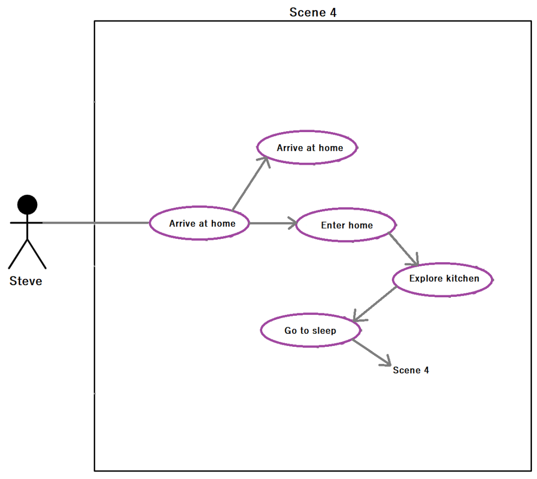
2.5.2. Use case model

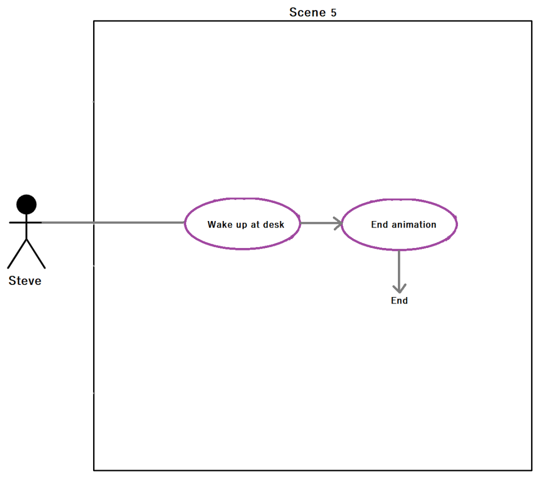
Christian Schäfer
Timo Skrobanek
Andrei Costeniuc











UseCase Description:

The main character is called Steve and he's a divorced man who has psychological problems.

In the first scene he wakes up on his sofa and the player can interact in first person with items inside the house.
The player has to accomplish tasks before being able to continue with the story. He has to get dressed, clean the kitchen, pick up a letter laying on the table and finally leave the house and drive to the city to work by clicking on the house door.

For the second scene the player will be teleported into the office and is able to sit down at his desk.

As the third scene starts the player is now at his desk where he can start to work by clicking on the computer. That's also where Steve will fall asleep.

In the fourth scene the player first has to enter his house and follow a mostly strict order of tasks which trigger new events.
First he has to enter the house and then explore the kitchen where he will see various optical illusions and figures moving around the house.
He will then have to go into the bedroom where he goes to bed and wakes up at his desk again. This leads to scene 5.

The fifth scene only consists of a cut scene where it is shown that Steve wakes up on his desk while his boss is yelling at him. The cut scene ends in Credits and the game is finished.

2.5.3. Object model

2.5.3.1. Data dictionary

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

2.5.3.2. Class diagrams

2.5.4. Dynamic models

2.5.5. User-interface

2.6 Sources

3. Glossary

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

AB HIER SIND SACHEN DIE ALS NEUES APPENDIX SOLLEN

Usability-Test für Szene 1 und 2

Ziel des Tests:
Die Testpersonen sollen mögliche Schwierigkeiten in der Benutzererfahrung identifizieren, insbesondere in Bezug auf das Navigieren, Erkennen von Zielen und die intuitive Interaktion mit der Spielumgebung.

Anweisungen für die Testperson:
Bitte folge den Anweisungen Schritt für Schritt und beantworte die Fragen so genau wie möglich.

Abschnitt 1: Navigation und erste Szene

1. **Spielstart**
   - **Aufgabe:** Starte das Spiel und navigiere zur ersten Spielszene (Zuhause).
   - **Fragen:**
     - Gab es Schwierigkeiten beim Finden oder Starten der Szene? *[Ja/Nein]*
2. **Interaktion in der ersten Szene**
   - **Aufgabe:** Räume alle Items auf, die auf der Couch oder in der Küche liegen.
   - **Fragen:**
     - Konntest du alle Items leicht finden? *[Ja/Nein]*
     - Ist es offensichtlich, ob genügend Items aufgeräumt wurden? *[Ja/Nein]*

Abschnitt 2: Interaktion mit Objekten

1. **Anziehen der Kleidung**
   - **Aufgabe:** Ziehe deine Klamotten an und gehe zur Arbeit.
   - **Fragen:**
     - War der Kleiderschrank leicht zu finden? *[Ja/Nein]*
2. **Finden des Arbeitsplatzes in Szene 2**
   - **Aufgabe:** Im Büro, finde deinen Arbeitsplatz und setze dich hin.
   - **Fragen:**
     - War der Arbeitsplatz eindeutig und leicht zu finden? *[Ja/Nein]*

Christian Schäfer
Timo Skrobanek
Andrei Costeniuc

Bewertung der Benutzererfahrung

1. **Allgemeine Benutzererfahrung**
    - **Frage:** Auf einer Skala von 1 bis 10, wie würdest du die Benutzererfahrung bewerten (Wie einfach ist es, sich im Spiel zurechtzufinden)?
      [1 = sehr schwierig, 10 = sehr einfach]
2. **Verbesserungsvorschläge**
    - **Frage:** Was würdest du am Spiel ändern, um die Benutzererfahrung zu verbessern?
      [Bitte erläutere]

Hinweise für die Durchführung

- Der Test wird per Google Forms durchgeführt und die Auswertung wird als PDF angehängt.

Siehe Google Forms

//TODO Google Forms ausfüllen und Statistiken als PDF einfügen