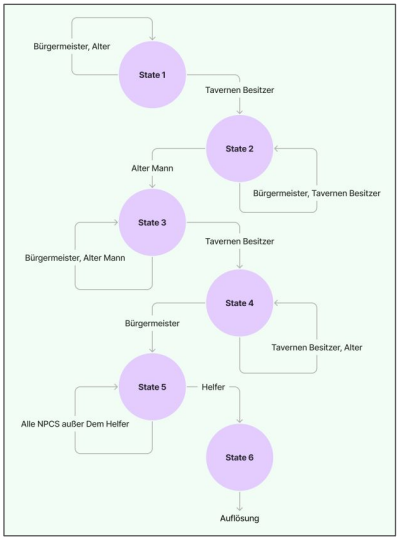
Audit 3

Visual Computing - Entwicklungsprojekt WS24/25
Lukas Diekmann, Amin Merzouk und Morris Schacht
Team Anya

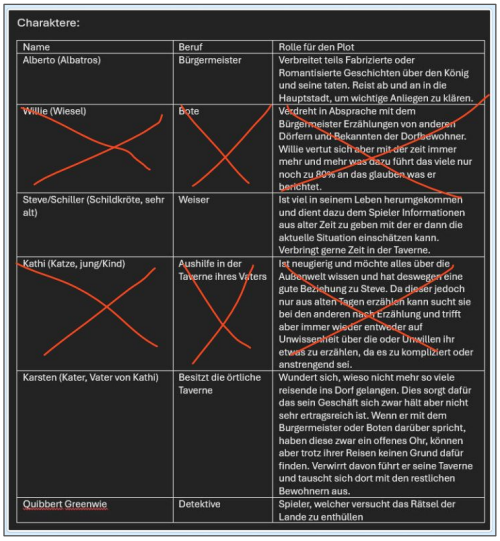
Iteriertes

Modelle, Rapid Story

Modelle



Game States



Charaktere:

Name	Beruf	Rolle für den Plot
Alberto (Albatros)	Bürgermeister	Verbreitet teils Fabrizierte oder Romantisierte Geschichten über den König und seine taten. Reist ab und an in die Hauptstadt, um wichtige Anliegen zu klären.
Willie (Wiesel)	Bote	Verdreht in Absprache mit dem Bürgermeister Erzählungen von anderen Dörfern und Bekannten der Dorfbewohner. Willie vertut sich aber mit der zeit immer mehr und mehr was dazu führt das viele nur noch zu 80% an das glauben was er berichtet.
Steve/Schiller (Schildkröte, sehr alt)	Weiser	Ist viel in seinem Leben herumgekommen und dient dazu dem Spieler Informationen aus alter Zeit zu geben mit der er dann die aktuelle Situation einschätzen kann. Verbringt gerne Zeit in der Taverne.
Kathi (Katze, jung/Kind)	Aushilfe in der Taverne ihres Vaters	Ist neugierig und möchte alles über die Außenwelt wissen und hat deswegen eine gute Beziehung zu Steve. Da dieser jedoch nur aus alten Tagen erzählen kann sucht sie bei den anderen nach Erzählung und trifft aber immer wieder entweder auf Unwissenheit über die oder Unwillen ihr etwas zu erzählen, da es zu kompliziert oder anstrengend sei.
Karsten (Kater, Vater von Kathi)	Besitzt die örtliche Taverne	Wundert sich, wieso nicht mehr so viele reisende ins Dorf gelangen. Dies sorgt dafür das sein Geschäft sich zwar hält aber nicht sehr ertragsreich ist. Wenn er mit dem Burgermeister oder Boten darüber spricht, haben diese zwar ein offenes Ohr, können aber trotz ihrer Reisen keinen Grund dafür finden. Verwirrt davon führt er seine Taverne und tauscht sich dort mit den restlichen Bewohnern aus.
Quibbert Greenwie	Detektive	Spieler, welcher versucht das Rätsel der Lande zu enthüllen

Charaktere LvL1

Zur Umsetzung der PoCs und zur generellen Weiterentwicklung unser Idee, wurde die Modellierung der Game States und die involvierten Charaktere für das erste Level iteriert und an manchen stellen Vorläufig gekürzt. Erst wurden die Charakter für das erste Level etwas gekürzt um möglichst qualitativ zu garantieren das unsere Idee sich umsetzen lässt. Darauf aufbauend kann man in Zukunft dann aufbauen und die Charaktere und die Komplexität erweitern.

PoCs

Übersicht, In-Engine, Code

Was wurde umgesetzt

POC's

Speichern:
- Daten werden im JSON übergeben

Dialogsystem:
- Basiert auf dem Endlichen Automaten

Levelauswahl:
- Funktionale UI wurde umgesetzt

Notizbuch
- Geschehene Dialoge werden wiedergegeben

Nach dem letzten Audit wurden folgende POCs umgesetzt:

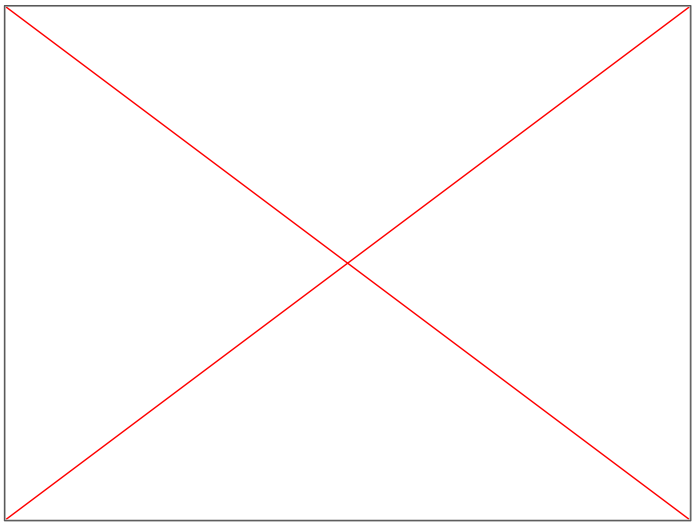
Speichern:
Das Speichern der Daten erfolgt über JSON. Im jetzigen Entwicklungsstand lassen sich die zu speichernden Daten flexibel erweitern. Der Code wurde so aufgebaut, dass jederzeit entschieden werden kann, wann ein Speichervorgang durchgeführt wird.

Dialogsystem:
Im Audit hatten wir ursprünglich geplant, ein vorgefertigtes Dialogsystem zu verwenden. Da jedoch das Risiko des Brut-Forcings von Dialogen diskutiert wurde, entschieden wir uns stattdessen, ein eigenes Dialogsystem zu entwickeln. Die Logik basiert auf der Automatentheorie. Die Dialoge können in einem JSON gespeichert werden und werden bei einer Interaktion mit einem NPC auf Grundlage eines akzeptierten Wortes aufgerufen.

Levelauswahl:
Über die Levelauswahl können unterschiedliche Szenen aufgerufen werden. Diese können unabhängig voneinander gestaltet werden und sind über Einträge im System flexibel integrierbar.

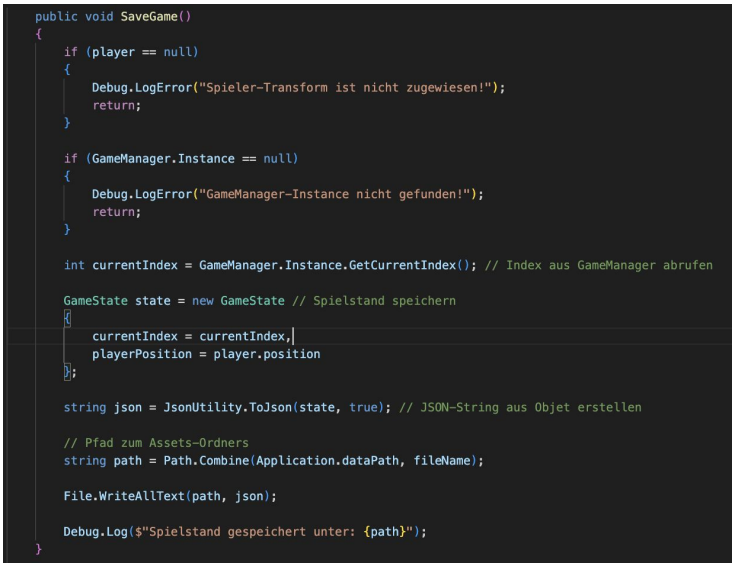
Notebook:
Das Notebook gibt alle vergangenen Dialoge wieder. Dabei orientiert sich die Reihenfolge sowohl an den Wörtern selbst als auch an deren Position im Dialog.

In Engine



In diesem Video sieht man wie das Dialogsystem funktioniert. Um brute forcing zu vermeiden verwenden wie die Theorie des Endlichen Automaten in unserem GameManager. Der GameManager gilt als Observer der sichergeht das das Wort eingehalten wird. Die einzelnen Buchstaben im Wort stehen für die NPC die nach der Reihenfolge des Wortes angesprochen werden können. Weicht der Spieler davon ab wird ein Standarddiaolog ausgegeben. Spricht der Spiele den richtigen NPC an wird das Wort vom letzten akzeptierten Buchstaben weitergeführt.

PoC Speichern



```
public void SaveGame()
{
    if (player == null)
    {
        Debug.LogError("Spieler-Transform ist nicht zugewiesen!");
        return;
    }

    if (GameManager.Instance == null)
    {
        Debug.LogError("GameManager-Instance nicht gefunden!");
        return;
    }

    int currentIndex = GameManager.Instance.GetCurrentIndex(); // Index aus GameManager abrufen

    GameState state = new GameState // Spielstand speichern
    {
        currentIndex = currentIndex,
        playerPosition = player.position
    };

    string json = JsonUtility.ToJson(state, true); // JSON-String aus Objet erstellen

    // Pfad zum Assets-Ordners
    string path = Path.Combine(Application.dataPath, fileName);

    File.WriteAllText(path, json);

    Debug.Log($"Spielstand gespeichert unter: {path}");
}
```


Im GameSaver werden die zu speichernden Parameter definiert und in einer JSON-Datei gespeichert. Der Code überprüft zunächst, ob die notwendigen Komponenten, wie der Spieler-Transform und die GameManager-Instanz, vorhanden sind. Wenn diese fehlen, wird eine Fehlermeldung ausgegeben und der Speichervorgang abgebrochen.

Der aktuelle Index, der den Fortschritt im Spiel repräsentiert, wird über den GameManagers aufgerufen. Danach wird ein GameState-Objekt erstellt, in dem sowohl der aktuelle Index als auch die Position des Spielers gespeichert werden.

Das GameState-Objekt wird anschließend in einen JSON-String umgewandelt, um sie in einer JSON-Datei zu speichern.

PoC Dialogsystem

```
public string GetNextDialogue(char npcLetter)
{
    // Prüfe, ob der NPC in der Reihenfolge korrekt ist
    if (currentIndex < targetWord.Length && targetWord[currentIndex] == npcLetter)
    {
        // Prüfe, ob Dialoge für diesen NPC existieren
        if (npcDialogues.ContainsKey(npcLetter))
        {
            List<string> dialogues = npcDialogues[npcLetter];

            // Prüfe, ob es noch einen Dialog gibt
            if (npcDialogueIndex[npcLetter] < dialogues.Count)
            {
                string dialogue = dialogues[npcDialogueIndex[npcLetter]];
                npcDialogueIndex[npcLetter]++; // Fortschritt für diesen NPC
                currentIndex++; // Fortschritt im Wort
                return dialogue;
            }
            else
            {
                return $"Keine weiteren Dialoge für NPC {npcLetter}.";
            }
        }
        else
        {
            return $"Dialoge für NPC {npcLetter} nicht gefunden.";
        }
    }
    else
    {
        return $"NPC {npcLetter} ist nicht der nächste in der Reihenfolge.";
    }
}
```


Das Dialogsystem basiert auf der Automatentheorie und überprüft das vorgegebene Wort (das Zielwort) die die Interaktion Reihenfolge bestimmt. Der Code ist zuständig dafür, dass Dialoge nur in der richtigen Reihenfolge und bei passenden Bedingungen abgerufen werden.

Die Methode GetNextDialogue nimmt einen Buchstaben (npcLetter) entgegen, der den aktuellen NPC repräsentiert. Zunächst wird geprüft, ob der Buchstabe des NPCs mit dem aktuellen Buchstaben des Zielworts übereinstimmt. Nur wenn das der Fall ist, wird die Logik für den Dialogfortschritt ausgeführt.

Wenn Dialoge für diesen NPC existieren, wird überprüft, ob weitere Dialoge verfügbar sind. Ist das der Fall, wird der nächste Dialog aus der Liste abgerufen und sowohl der Dialogindex für diesen NPC als auch der allgemeine Fortschritt im Zielwort erhöht. Andernfalls gibt die Methode eine Meldung zurück, dass keine weiteren Dialoge für den NPC vorhanden sind.

Falls es für den NPC keine Dialoge gibt oder der Buchstabe nicht mit der Reihenfolge des Zielworts übereinstimmt, wird eine entsprechende Fehlermeldung ausgegeben.

PoC Level auswahl

```
public class MainMenuScript : MonoBehaviour
{
    2 Verweise
    private UIDocument _document;

    2 Verweise
    private Button _button;

    0 Verweise
    private void Awake()
    {
        _document = GetComponent<UIDocument>();

        _button = _document.rootVisualElement.Q("level1") as Button;
        _button.RegisterCallback<ClickEvent>(OpenLevel);
    }



    1 Verweis
    public void OpenLevel(ClickEvent evt)
    {
        SceneManager.LoadScene("SampleScene");
    }
}
```


DIe Levelauswahl wurde mit Hilfe des UIToolkits innerhalb von Unity erstellt. Im Anschluss wurde ein Script implementiert, welches die Button um on-click Events erweitert. Zum aktuellen Stand der Entwicklung wurde nur der “Level 1- Button” mit einer weiterleitung auf das erste Level versehen.
Die Weiterleitung funktioniert über den SceneManager, welcher es ermöglicht mit der LoadScene() Funktion zwischen einzelnen Szenen hin und her zu manövrieren. Dazu muss lediglich der Name der Szene übergeben werden.

PoC Notebook

```
// Gehe die NPCs im targetWord durch und gebe nur den Dialog für die aktuelle Stelle im Wort aus
for (int i = 0; i < currentIndex; i++) // Bis zum aktuellen Index im targetWord
{
    char npcLetter = targetWord[i]; // Bestimmen, welcher NPC an der Reihe ist
    if (npcDialogues.TryGetValue(npcLetter, out var dialogues))
    {
        // Sicherstellen, dass wir den Dialog an der aktuellen Stelle bekommen
        if (i < dialogues.Count)
        {
            Debug.Log($"NPC {npcLetter}: {dialogues[i]}");
        }
        else
        {
            Debug.Log($"NPC {npcLetter}: Kein Dialog gefunden.");
        }
    }
    else
    {
        Debug.Log($"NPC {npcLetter} hat keine Dialoge.");
    }
}
```


Das Notebook ist dazu da alle Dialoge bis zum aktuellen Fortschritt im Zielwort (targetWord) auszugeben. Dafür wird jeder Buchstabe im Zielwort bis zum aktuellen Index (currentIndex) durchlaufen, um den entsprechenden NPC zu bestimmen.

Für jeden NPC wird geprüft, ob Dialoge vorhanden sind. Ist das der Fall, wird der passende Dialog an der jeweiligen Stelle ausgegeben. Wenn es für den aktuellen NPC keinen Dialog an der Stelle gibt, wird dies entsprechend gemeldet. Existieren für den NPC überhaupt keine Dialoge, wird ebenfalls eine Fehlermeldung ausgegeben.

Aussichten

Verbesserungen, Ziele, Zeitplan A4

Verbesserungen

Dialogsystem:
- optimieren des Algorithmus

Story:
- Ausarbeitung der Charaktere und ihrer involvierung
- Glaubwürdige und nicht Hochkomplexe Charaktere

Dialogbaum erweitern:
- Aufbesserung des Handlungsstrangs
- Bewahrung einer verständlichen Geschichte

Eine Aussicht ist die Optimierung des Dialogsystems, spezifisch das Auslesen des JSON. Der Algorithmus zum Auslesen kann simpler gestaltet werden, um das Einfügen der Dialoge übersichtlicher zu machen. Zusätzlich kann man die Geschichte und ihre Charaktere einerseits noch Erweitern und dabei aber trotzdem einen übersichtlichen Handlungsstrang bewahren. Beispielsweise durch das hinzufügen von mehr oder komplexeren Charakteren und einer erweiterung des Dialogbaums könnte man dies gut umsetzen und in den Prototyp einbauen.

Ziele

- Visuelle überarbeitung der Umgebungen und Charaktere
- Durchdachtes und sinnvolles Leveldesign (hilft Spielern)
- Erweiterung der dialogstruktur (eventuell mehrer optionen)
- Erweiterung der Geschichte (konzeptuell 2 Szenario)

Diese Folie dient der Darstellung von persönlichen zielen die man noch innerhalb des Projekts Zeitraumes plant umzusetzen. Diese können kleinen Punkte wie dem einfügen von sounds bis hin zu größeren Punkten wie ein komplexes dialogsystem sein. Dabei ist zu beachten, dass diese einerseits nicht alle umgesetzt werdem müssen und das manche sich auch nicht im vorgegebenen Zeitraum umsetzen lassen. Trotzdem wollten wir diese hier einbauen um jedem zu ermöglichen vorhandene Ambitionen vorzustellen.

Zeitplan Audit 4

Hauptaufgabe	W1	W2	W3	W4	W5	Wer?
Audit 4						Alle
Leveldesign						Morris, Amin
Dialogsystem						Morris
Levelauswahl						Lukas
Models						Amin
Poster						Alle
Reflektion						Alle
Repo-Cleanup						Alle
Story						Amin, Lukas
Feedback						Alle
Spaß Haben :)						Alle

Für Audit 4 steht die Finale iterierung und weiterentwicklung unseres Rapid Prototypes hin zu einem funktionalen Prototypes im Fokus. Dafür sind sowohl feinschliffe im Code, der dargestellten Inhalte als auch der visuellen Darstellung von nöten. Um dies gescheit umsetzen zu können planen wir grob mit diesem Zeitplan. Dieser ist wie vieles subject to change und wird sich im verlauf der wochen nach A3 noch verändern.