Meine Titelseite

Unterüberschrift oder Autor

Datum

Kurzfassung

Die App-Programmierung ist ein Schlüsselelement in der digitalen Landschaft, das die Entwicklung von Anwendungen für mobile Geräte wie Smartphones und Tablets umfasst. Mit fundierten Kenntnissen in Programmiersprachen wie Java, Swift und Kotlin sowie in Frameworks wie React Native oder Flutter können Entwickler benutzerfreundliche und leistungsstarke Anwendungen erstellen. Eine gut gestaltete Benutzeroberfläche und eine reibungslose Benutzererfahrung sind entscheidend. Aktuelle Trends umfassen KI-Integration, AR und VR, Blockchain und IoT, die neue Möglichkeiten für innovative und interaktive Apps eröffnen. Die App-Programmierung bietet eine spannende Chance, kreative Ideen in funktionale Anwendungen umzusetzen, die das Leben vieler Menschen bereichern können.

Abstract

App development is a cornerstone of the digital landscape, encompassing the creation of applications for mobile devices such as smartphones and tablets. With proficient knowledge in programming languages like Java, Swift, and Kotlin, as well as frameworks like React Native or Flutter, developers can craft user-friendly and powerful applications. A well-designed user interface and smooth user experience are crucial. Current trends include AI integration, AR and VR, Blockchain, and IoT, which offer new avenues for innovative and interactive apps. App development presents an exciting opportunity to translate creative ideas into functional applications that can enrich the lives of many.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

St. Pölten, 31.Jänner 2025

Danksagung

Mein herzlicher Dank gilt meinem Technologiemangement-Lehrer für seine Unterstützung bei der Erstellung der LaTeX-Vorlage. Außerdem bedanke ich mich bei meinen Eltern und meiner Tante für das Korrekturlesen.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

In der heutigen digitalen Ära spielt die Entwicklung von Anwendungssoftware eine immer wichtigere Rolle, insbesondere im Kontext der zunehmenden Digitalisierung verschiedener Aspekte des täglichen Lebens.

1.1 Bestehende Software

Apps können durch die Verwendung verschiedener Programmiersprachen wie Java, Kotlin für Android oder Swift für iOS entwickelt werden. Die Entwicklungsumgebung bietet Werkzeuge wie Android Studio für Android oder Xcode für iOS, die Entwicklern helfen, Benutzeroberflächen zu gestalten, Funktionalitäten zu implementieren und mit APIs zu interagieren. Anschließend wird die App auf einem Emulator oder einem physischen Gerät getestet und schließlich in den jeweiligen App Stores veröffentlicht. Selbst Alan Turing (vgl. [Tur36]) konnte das nicht ahnen.

1.1.1 Android

Android ist ein Betriebssystem für mobile Geräte, das von Google entwickelt wird (siehe [and]). Es basiert auf dem Linux-Kernel und bietet eine offene Plattform für Entwickler. Android ermöglicht die Entwicklung vielfältiger Anwendungen, von Spielen über Produktivitäts-Apps bis hin zu sozialen Medien. Der Google Play Store bietet eine riesige Auswahl an Apps für Android-Geräte.

KAPITEL 2

Theorie

Auch hier stehen wieder ein paar Zeilen Text.

2.1 Über Anroid Studio

Apps können durch die Verwendung verschiedener Programmiersprachen wie Java, Kotlin für Android oder Swift für iOS entwickelt werden. Die Entwicklungsumgebung bietet Werkzeuge wie Android Studio für Android oder Xcode für iOS, die Entwicklern helfen, Benutzeroberflächen zu gestalten, Funktionalitäten zu implementieren und mit APIs zu interagieren. Anschließend wird die App auf einem Emulator oder einem physischen Gerät getestet und schließlich in den jeweiligen App Stores veröffentlicht.

2.2 Aufzählungen

Auch bei Aufzählungen sollte man entsprechende Befehle verwenden.

- **Lehrer:** Das ist jetzt nur ein Demotext, der über eine Zeile geht. Hier wird ersichtlich, dass man sich nicht um Abstände etc. kümmern muss.
- Schüler: Bei diesem Aufzählungspunkt ist das erste Worte nun nicht fett geschrieben.
- Räume

Natürlich kann man auch nummerierte Aufzählungen einfügen:

1. TMAN
2. HAK
3. St. Pölten

KAPITEL 3

Programmierung der App

3.1 Struktur

Im folgenden Abschnitt wird die Generelle Klassenstruktur vorgestellt

```
class
{
 public:
      // Oefentliche Methoden
      void Initialize();
      void Update();
      void Draw();
 private:
      // Private Methoden
 public:
      // Oeffentliche Variablen
 private:
      // Private Variablen
};
```

Listing 3.1: Klassenstruktur

3.2 Schreibstil

Das gesamte Programm wurde mit einem einheitlichen Stil geschrieben, dieser wird in diesem Abschnitt vorgestellt.
Methoden beginnen immer mit einem Großbuchstaben, so werden diese klar von Variabeln getrennt.

```
class
{
```

```
      public:
4           void Beispielsmethode();
      };
```

Listing 3.2: Methoden

Variablen welche member einer Klasse sind werden mit einem preafix gekennzeichnet und kleingeschrieben

```
1     class
      {
3           // Methoden
            ...
5     public:
            // Oeffentliche Variablen werden mit dem Praefix p_
               makiert
7     private:
            // Private Variablen werden mit dem Praefix m_ makiert
9     };
```

Listing 3.3: membervariabeln

Lokale Variabeln zu gänze kleingeschrieben und erhalten keinen Praefix.

```
1     class
      {
3     public:
            void Beispielsmethode()
5           {
                  int beispielsvariabel;
7           }
            ...
9           // Membervariabeln
      };
```

Listing 3.4: lokale variabeln

Bei Parametern wird der erste Buchstabe großgeschrieben, dadurch können Parameter von Member- und Lokalvariabeln unterschieden.

```
      class
2     {
            public:
4           void Beispielsmethode(int Beispielsparameter)
            {
6                 int beispielsvariabel = Beispielsparameter;
            }
8           ...
            // Membervariabeln
10    };
```

Listing 3.5: Parameter

3.3 Generieren der Karte

Jedes Spiel braucht eine Karte, in diesem Abschnitt werden wir uns Schrittweise die Implementierung der Map Klasse anschauen.

```
class Map
{
public:
      Map()
      {}

      void Initialize();
      void Generate();
      void Draw();
      ...
};
```

Listing 3.6: Map

Die Methode İnitialize"hat den Zweck member Variablen einen Wert zuzuweisen und gegenbenfalls in der Klasse enthaltene Objekte zu Initalisieren.

```
      ...
      void Initialize();
      ...
```

Die Idee ist es die Karte als 2-Demensionales Liste aus "Tiles"darzustellen, vorab definieren wir also die Größe Karte, sowie die Größe der "Tiles".

```
      ...
      void Initialize(const sf::Vector2u& tilesize, const int& width,
          const int& height);
      ...
```

Diese Werte sollen in der Klasse abgespeichert werden. Daher definieren wie nun die drei Member "tilesize", "width", und "height".

```
      ...
      private:
            sf::Vector2u m_tilesize;
            int m_width;
            int m_height;
      ...
```

Anschließend weisen wir den Membern den jeweiligen Parametern zu.

```
      ...
      void Initialize(const sf::Vector2u& tilesize, const int& width,
          const int& height)
      {
            m_tileSize = tilesize;
            m_width = width;
            m_height = height;
```

```
    }
    ...
```

Listing 3.7: Map::Initialize()

Die Methode ist vorerst fertiggestellt, nächster Schritt ist nun die Generierung. Wie schon erwähnt ist die Idee, die Karte als 2-Demensionale Liste bestehend aus "Tiles"darzustellen. Ähnlich wie das auch andere Spielen machen. (Beispiel Anführen) Die einzelnen Tiles sollen Informationen über Textur, Position und deren ID beeinhalten. Hierfür verwenden wir den Datentyp SStruct". Structs unterscheiden sich in c++ nicht wesentlich von Klassen, jedoch werden wir aus Stilgründen den Typ Struct verwenden. Wichtig: Structs sollen jediglich informationen beeinhalten und üben sonst keine Funktion aus.

```
    struct Tile
    {
        ...
        unsigned int tile_ID;
        sf::Vector2f tile_position;

        sf::IntRect* tile_texRect;
        sf::Sprite tile_sprite;

    };
```

Der Konstruktor wird definiert, wir erfassen die ID, die Position und die Textur.

```
    ...
    Tile(const unsigned int& ID, sf::IntRect* TextureRectangle,
       const sf::Sprite &Sprite, const sf::Vector2f& TilePosition)
    ...
```

Die die Member werden mit den Parametern initialsiert

```
    ...
    : tile_ID(ID), tile_texRect(TextureRectangle), tile_sprite(
       Sprite), tile_position(TilePosition)
    ...
```

Der Textur wird die Position und Texture Rectangle zugewiesen.

```
    ...
    {
        tile_sprite.setPosition(tile_position);
        tile_sprite.setTextureRect(*tile_texRect);
    }
    ...
```

Alle Teile zusammen ergeben einen Funktionierenden Datentyp in dem wir alle notwendigen Information Speichern.

```
    struct Tile
    {
```

```
            Tile(const unsigned int& ID, sf::IntRect*
                TextureRectangle, const sf::Sprite &Sprite, const sf::
                Vector2f& TilePosition)
4           : tile_ID(ID), tile_texRect(TextureRectangle),
                tile_sprite(Sprite), tile_position(TilePosition)
            {
6                   tile_sprite.setPosition(tile_position);
                    tile_sprite.setTextureRect(*tile_texRect);
8           }
            ...
10      };
```

Doch noch ein Schritt fehlt um mit der Generierung zu Starten. Da die Karte in der Klasse abgespeichert werden soll fügen wir den entsprechenden Member hinzu. Hiefür verwenden wir den Typ std::vector. Dieser fungiert als herkömmliche liste, die aber nicht statisch initalisiert werden muss wie es beispielsweise bei Tile arr[]; der Fall wäre. Später sollen andere Klassen noch auf die Karte zugreifen also schreiben wir sie als public.

```
        ...
2       std::vector<std::vector<Tile>> p_tileMap;
        ...
```

Listing 3.8: Tile

Nach dem alle vorbereitungen getroffen worden sind, können wir mit der Generierung beginnen.

```
1       ...
        void Generate();
3       ...
```

Zunächst wollen wir p_tilemap mit Tiles füllen. Dafür iterieren wir entlang der x-achse und erstellen einen std::vector den wir mit den noch leeren Tiles füllen. Anchließend wird der gefüllte std::vector an p_tilemap angefügt.

```
1   void Generate()
    {
3       for (int i = 0; i < m_width; i++)
        {
5           std::vector<Tile> tileMap_row;

7           for (int j = 0; j < m_height; j++)
            {
9
                    tileMap_row.push_back(tile);
11          }

13          p_tileMap.push_back(tileMap_row);
        }
15  }
```

An dieser Stelle verwenden wir Prozedurale Generation - Diese ist zu diesem Zeitpunkt nicht implementiert oder Erklärt. Daher machen wir uns eine Mentale Notiz und kommen später wieder zurück. Die grundsätzliche Idee ist es eine 2-Demensionale Liste mit Semi-Zufälligen Werten zu füllen. Jeder dieser Werte stellt ein Bestimmtes Biom da. Demnach laden wir die textur welche dem Biom entspricht. Vorerst schreiben wir also:

```
...
int biome = // Das Biom an der stelle [i][j]
...
```

Wir verwenden das Biom um die richtige Textur, sowie die richtige Position zu finden.

```
...
sf::IntRect texRect(m_tileSize.x * biome,
                    m_tileSize.y * biome,
                    m_tileSize.x, m_tileSize.y);
sf::Vector2f tilePos(m_tileSize.x * i, m_tileSize.y * j);
...
```

Anschließend verwenden wir den Konstrukter des Tile-Struct um dem Tile die Daten zuzuweisen.

```
      ...
      Tile tile(index, biome, &texRect, m_tilesprite, tilePos);
      ...
```

Die vollständige Methode:

```
void Generate()
{
 for (int i = 0; i < m_width; i++)
 {
  std::vector<Tile> tileMap_row;
  for (int j = 0; j < m_height; j++)
  {
   int biome = // Das Biom an der stelle index

   sf::IntRect texRect(m_tileSize.x * biome,
                       m_tileSize.y * biome,
                       m_tileSize.x, m_tileSize.y);

   sf::Vector2f tilePos(m_tileSize.x * i, m_tileSize.y * j);

   Tile tile(index, biome, &texRect, m_tilesprite, tilePos);

   tileMap_row.push_back(tile);
  }

  p_tileMap.push_back(tileMap_row);
 }
}
```

Listing 3.9: Map::Generate()

Nachdem wir Initalize() und Generate() bereits erstellt haben, fehlt uns nur noch Draw().

```
1       ...
        void Draw();
3       ...
```

Vorerst hat die Methode nur die Aufgabe alle Texturen der Tiles zu zeichnen. Das problem ist hierbei aber das wir gleich die Ganze Map rendern, das wiederrum führt zu problemen mit der Performance. Es sollte also nur ein Teil bzw. "Chunk"der Map gerendert werden.

```
1       void Draw(sf::RenderWindow& Window)
        {
3               for(auto& i : p_tilemap){
                        for(auto& j : i){
5                               window.draw(j.tile_sprite);
                        }
7               }
        }
```

Die Map-Klasse ist Fertig. (Vorerst) Auf der TO-DO-List: Prozedurale Generation mit Perlin Noise und Chunk rendering

Listing 3.10: Map::Draw()

3.4 Der Spieler

Erste uebersicht der Klasse:

```
class Player
{
 public:
      void Initalize();
      void Update();
      void Draw();

 private:
      void MovePlayer();
};
```

Wir beginnen mit der Methode Initlize(), zweck der Methode sollte zu diesem Punkt schon klar sein. Wir stellen uns also die Frage welche Daten die Klasse beeinhalten Soll. fuer dieses Projekt werden wir folgende verwenden: Geschwindigkeit, Lebenspunkte, Position, die Hitbox* und die Textur. als public schreiben wir die Hitbox, Position und Lebenspunkte. Der Rest wird als privat geschrieben.

```
      ...
      void Initalize();
      ...
```

```
      ...
public:
      sf::RectangleShape p_hitbox;
      int    p_health;
private:
      sf::Sprite  m_sprite;
      sf::Vector2f m_position;
      float   m_speed;
      int movementIndicator = 0; // wird spaeter erklaert
```

Nun fuegen wir die notwendigen hinzu und initalisieren die Member.

```
...
void Initalize(Textureholder& Textures, const float& Speed, const int
   & Health, const sf::Vector2f& Position);
...
```

Das für dieses Projekt verwendete Spieler-Sprite hat eine größe von 64x64 daher setzten wir in sf::IntRect die breite sowie die höhe auf 64 pixel. Die hitbox soll der Spritegröße entsprechen und bekommt daher den selben wert zugewiesen. Wir verwenden den Textureholder* um auf die Spieler Textur zuzugreifen.

```
      ...
      m_sprite.setTexture(Textures.Get(Textures::ID::Player));
      m_sprite.setTextureRect(sf::IntRect(0, 0, 64, 64));
      m_sprite.setPosition(Position);

```

```
      p_hitbox.setSize(sf::Vector2f(64, 64));
      ...
```

Die Update Methode der Spieler klasse soll die Spielerbewegungen festhalten, daher verändern wir die Position um die Geschwindigkeit * Deltatime gleichzeitg verschieben wir den Ausschnitt der textur (TextureRectangle). Fue diesen Zweck definieren wir die Makros* UPWARDS, DOWNWARDS, LEFTWARDS und RIGHTWARDS. die Werte stellen die Position in der jeweiligen Sprites innerhalb der Textur da. Um festzustellen ob die Spielerposition sich mit einem Hinderniss überschneidet übergeben wir die Map-Klasse an die Methode. Zusätzlich passen wir die Hitbox an die positon des Spielers an.

```
      ...
      #define FORWARD 0
      #define LEFTWARD 1
      #define BACKWARD 2
      #define RIGHTWARD 3
      ...
```

```
      ...
      void Update(const float& Deltatime, MapGenerator &Map)
      {
            MovePlayer(Deltatime, Map);
            p_hitbox.setPosition(m_position);
      }
      ...
```

Definieren wir die MovePlayer Methode.

```
      ...
      private:
       void MovePlayer(const float &Deltatime, MapGenerator &Map);
      ...
```

Fuer jede der Richtung verwenden die von SFML bereitsgestellte Methode ßf::KeyBoard::isKeyPressed ". Diese registriert eingaben die über die Tastatur getaetigt werden.

```
      ...
      if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
      ...
```

wir legen die neue position fest. Da der Spieler in richtung oben laufen soll multiplplizieren wir die änderung an der y-achse mit -1. Das funktioniert weil der punkt (0,0) Oben-Links befindet. (Quelle)

```
      ...
      sf::Vector2f newposition = m_position + sf::Vector2f(0, -1) *
         m_speed * dt;
      ...
```

An dieser stelle ist anzumerken das es wohl bessere wege gaebe um das problem zu lösen. Da der Userinput bei jeder iteration der Update methode abgefragt wird, kommt es zu

mehreren hunderten abfragen pro sekunde. Da das sich nicht mit der Anfordung ein visuell stimulierendes Spiel zu programmieren deckt, verhindern wir dies in dem wir eine Membervariabel deklarieren welche bei jeder Iteration mitlaeuft. Die Werte 9 und 5 sind hier trivial.
Zuletzt weisen wir den Member m_position die neue position zu.

```
1       ...
        m_sprite.setPosition(newposition);
3
        m_sprite.setTextureRect(sf::IntRect((movementIndicator / 5) *
            SPRITEUNIT, FORWARD, SPRITEUNIT, SPRITEUNIT));
5       movementIndicator++;
        if (movementIndicator / MOVEMENT == 9) {
7             movementIndicator = 0;
        }
9


11      m_position = m_sprite.getPosition();
}
13 ...
```

Da wir verhindert müssen das der Spieler mit hindernissen kollidert ueberpruefen wir ob die Spielerpostion auf der Tilemap bereits belegt ist. Dauer definieren wir die methode YouShallPass.

```
1       ...
        bool YouShallPass(const sf::Vector2f& reisepass, MapGenerator &
            map) {
3             int x = reisepass.x / map.GetTileSize().x, y = reisepass.
                  y / map.GetTileSize().y;
              if (map.p_tileMap[x][y].occupied != true) {
5                   return true;
              }
7             return false;
        }
9       ...
```

Da mit der jetzigen programmierung der Spieler daran gehindert wäre durch gegner durchzulaufen schreiben wir uns noch die Hilfsfunktion ÏsEnemy"diese überprüft ob sich die positon mit einem gegner überschneiden würde. Wir verwenden wir den Member des Tile-Structs welche wir im Kapitel 3.3 bereits kennengelernt haben, occupationID.

```
1       ...
        bool IsEnemy(MapGenerator& map, const int& x, const int& y) {
3             Textures::ID type = map.p_tileMap[x][y].occupationID;
              switch (type)
5             {
                    case Textures::ID::Zombie:
7                   return true;
              }
```

```
9
            return false;
11      }
        ...
```

Anschließend fuegen wir der Abfrage in der YouShallPass Methode noch die IsEnemy option hinzu.

```
        ...
2   if (map.p_tileMap[x][y].occupied != true || IsEnemy(map, x,y))
        ...
```

Der Code-Block wird zusättzlich noch für die fälle S, A und D bzw. (0, 1), (-1, 0), (1, 0) wiederhohlt.

```
1       ...
        void MovePlayer(const float& dt, MapGenerator& map)
3       {
              if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
5                 sf::Vector2f newposition = m_position + sf::
                      Vector2f(0, -1) * m_speed * dt;

7                 if (YouShallPass(newposition, map)) {
                        m_sprite.setPosition(newposition);
9                       m_sprite.setTextureRect(sf::IntRect((
                            movementIndicator / MOVEMENT) * SPRITEUNIT
                            , FORWARD, SPRITEUNIT, SPRITEUNIT));
                        movementIndicator++;
11                      if (movementIndicator / MOVEMENT == 9) {
                              movementIndicator = 0;
13                      }
                  }
15
              }
17            ...

19            m_position = m_sprite.getPosition();
        }
21      ...
```

KAPITEL 4

Prozessmanagement

Da stehen wieder ein paar Zeilen. Dieser Text verweist sogar auf das Bild 4.1, welches einen Projektstrukturplan zeigt. Man kann mit LaTeXauch auf eine Subsection verweisen, wie etwa auf die folgende mit der Nr. 4.1 verweisen.

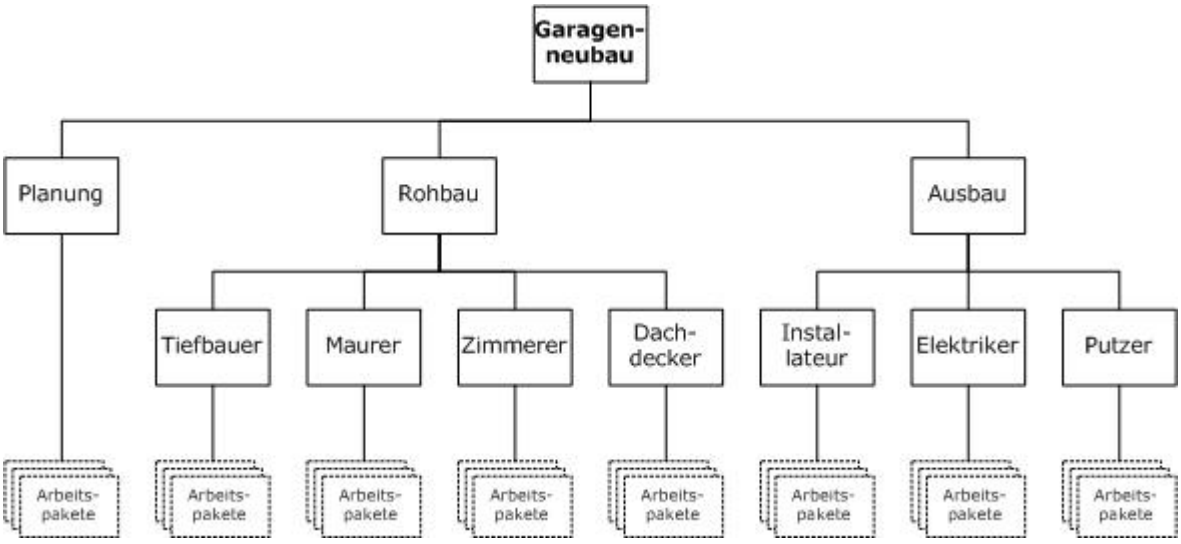

Abbildung 4.1: Demo-Projektstrukturplan

Tabelle 4.1: Demotabelle

Tätigkeit	Datum	Zeitdauer
A	B	C
D	E	F
D	E	F
D	E	F
D	E	F
D	E	F
D	E	F
D	E	F
D	E	F
D	E	F

4.1 Projektstrukturplan

4.2 Arbeitspakete

4.3 Projektwürdigkeitsanalyse

4.4 Projektdurchführbarkeitsanalyse

4.5 Meilensteinplan

4.6 Tätigkeitsliste - PersonA

Tabellen einfügen ist in LaTeXetwas schwieriger. Für das Grundgerüst bieten sich Online-Editoren wie etwa https://latex-editor.pages.dev/table/ an. Dabei ist dann der Tabular-Block zu kopieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[and] Download Android Studio & App Tools - Android Developers — developer.android.com. https://developer.android.com/studio. [letzter Zugriff: 28-02-2024].

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:345–363, 1936.