Schulische Projektdokumentation

Beteiligte Schüler: Maik Scheler, Julian Wening

Korrigierende Lehrer*innen: Rainhold Sauerbrey, Reinhard Fichtner, Mirjam Gehr-Nienhaus

Klasse: IFA12B

Zeitraum: 19.02.2024- 22.03.2024

Inhaltsverzeichnis

1 Mockup

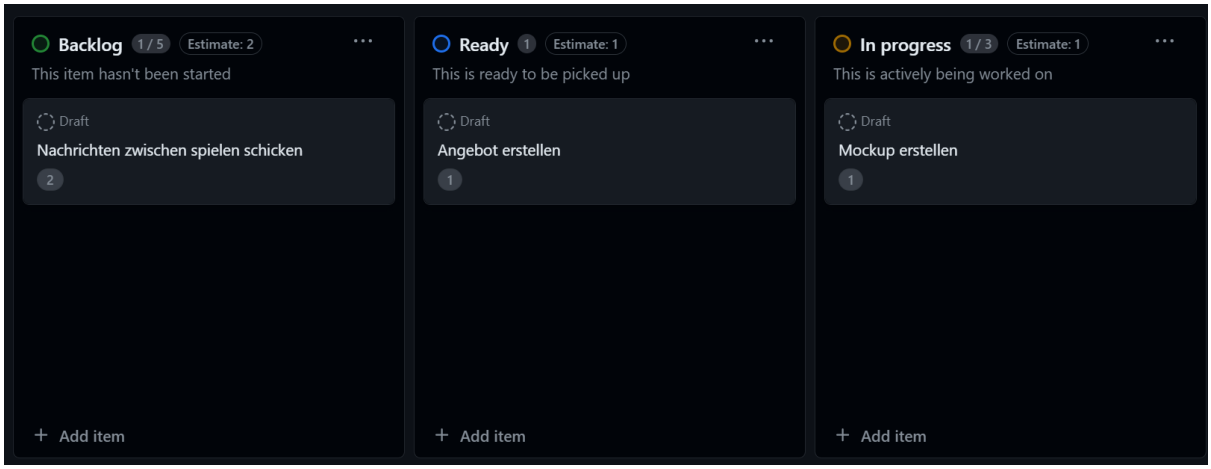
Bei dem Mockup in Abbildung 1 haben wir uns für ein relative simples design entschieden bei dem die Kartendecks den Primären Fokus haben. Durch das schlichte Design soll das Spiel intuitiv für jeden Nutzer sein.

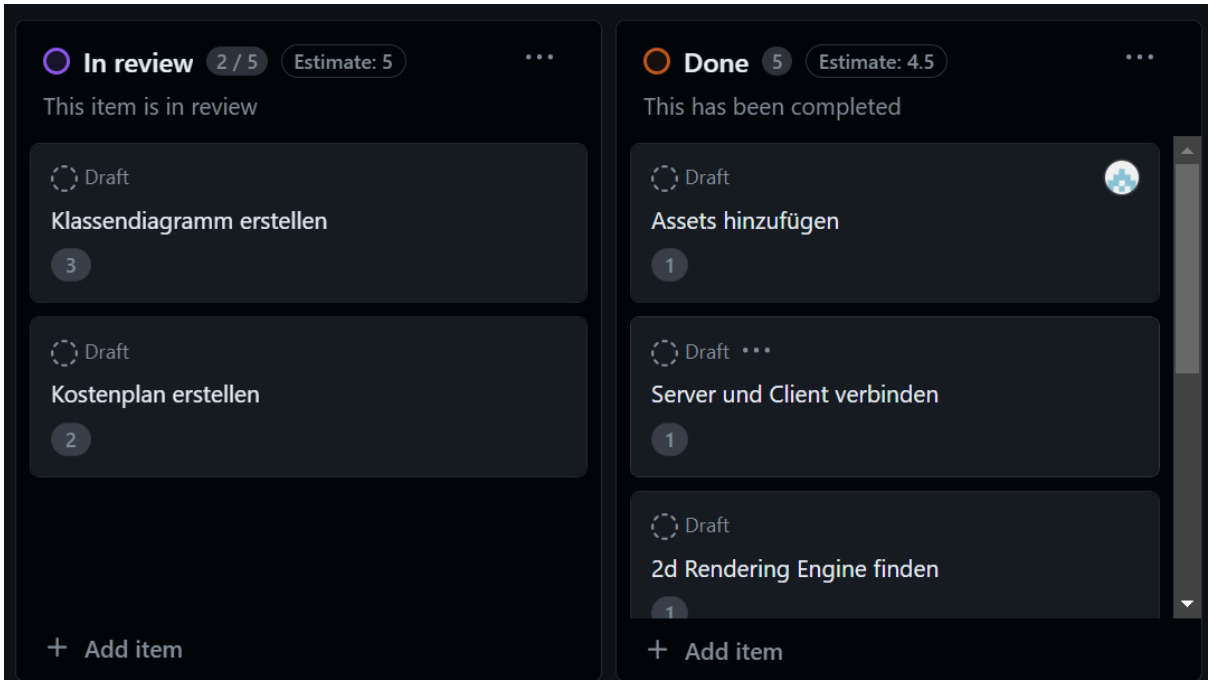
Die Karten auf der oberen Seite sind die Karten des Gegners, welche mit der Rückseite zu dir liegen. Somit kann man sehen wie viele Karten der Gegner hat.

Die Karten auf der unteren Seite gehören dem Spieler. Damit der Nutzer auf dem ersten blick sieht welche Karten er noch hat.

Abbildung 1

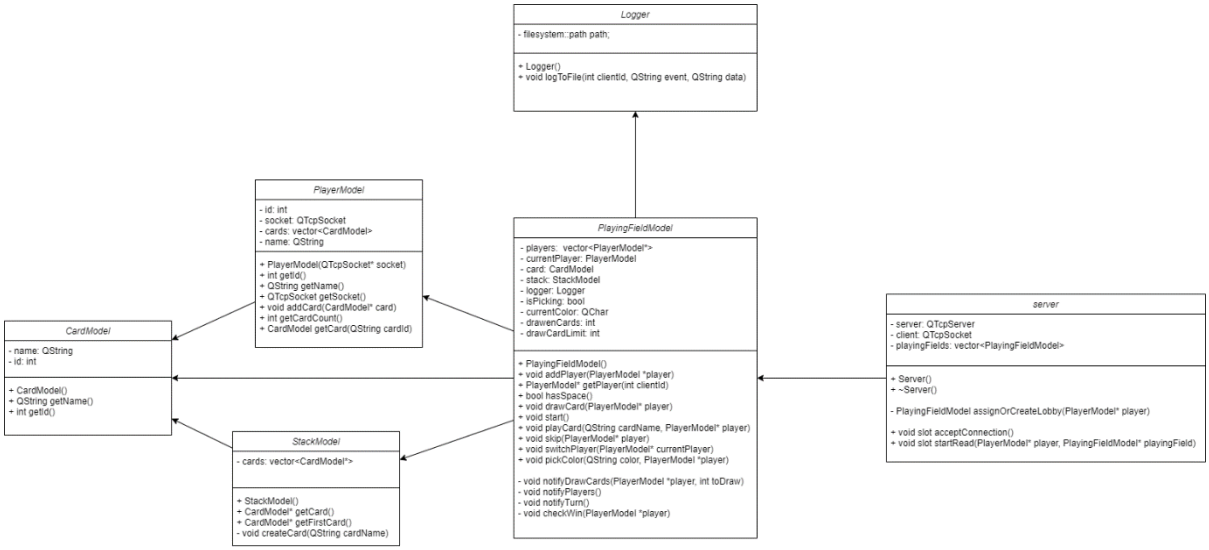
2 Kanban-Board

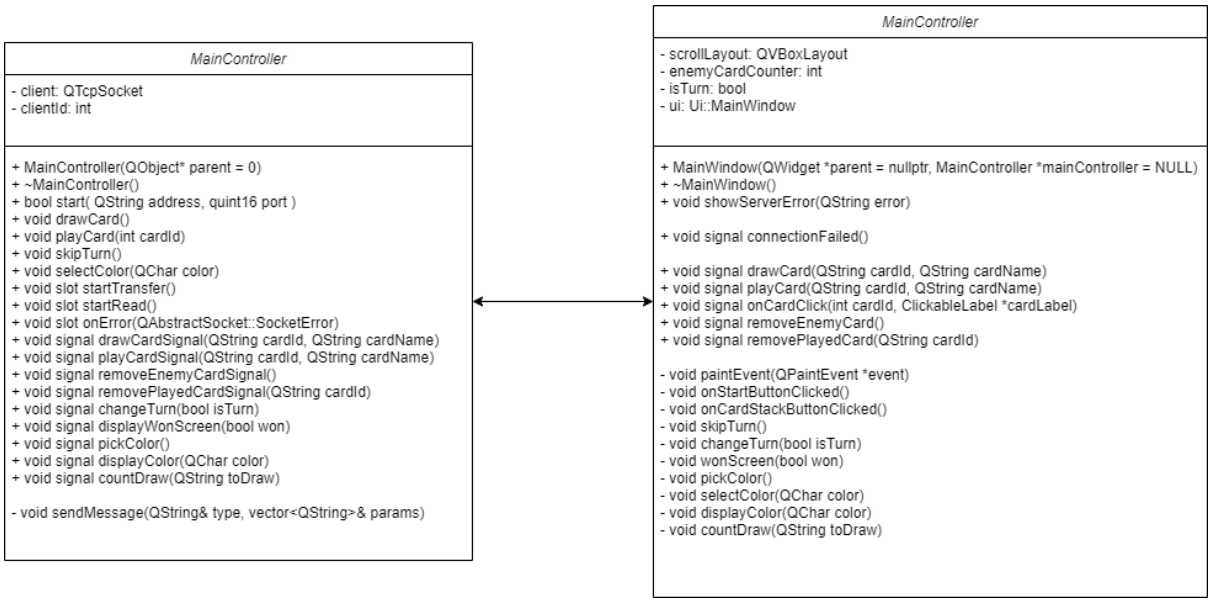




https://github.com/users/MaikScheler/projects/1/views/1

3 Klassen-Diagramm





4 Code-Ausschnitte

4.1 Client-seitig

```
/*
* Empfäng die Nachrichten vom Server und verarbeitet diese.
* Leitet die Nachrichten an die richtige funktion weiter
*/
void MainController::startRead(){
    char buffer[1024] = {0};
    QTcpSocket *sender = (QTcpSocket* ) QObject::sender();
    sender->read(buffer, sender->bytesAvailable());
    // Teilt die nachrichten in die einzelnen events auf
    QStringList responses = QString::fromLatin1(buffer).split(QRegExp("\n"));
    reverse(responses.begin(), responses.end());

    // geht alle Events die vom Server geschickt wurden durch
    for(QString _response : responses)
    {
        // Teil die nachricht in ihre einzelene bestandteile auf
        QStringList response = _response.split(QRegExp(":"));

        if(_response.size() == 0 || response.size() < 2)
        {
            continue;
        }

        QString type = response[0];
        QString _clientId = response[1];
        QString message = response.count() > 2 ? response[2] : "";

        //Empfangenen String zum Debuggen ausgeben (zum Testen)
        qDebug() << response;

        // Bekommt die seine Cilent id vom Server
        if(type == "connected")
        {
            clientId = _clientId.toInt();

        }
        // Event für die Karte die auf dem Stapel liegt
        else if(type == "card")
        {
            // Card id on index 3 and card name on index 2 (message)
            emit drawCardSignal(response[3], message);
        }
        // Event fürs Entfernen von Gespielten karten vom Gegener oder der eigenen
        else if(type == "play")
        {
            // Card id on index 3 and card name on index 2 (message)
            if(_clientId.toInt() != clientId)
            {
                emit removeEnemyCardSignal();
            } else {
                emit removePlayedCardSignal(response[3]);
            }

            emit playCardSignal(response[3], message);
        }
        // Event wer am zug ist
        else if (type == "turn") {
            int sendClientId = _clientId.toInt();
            bool turn = sendClientId == clientId;
            emit changeTurn(turn);
        }
        // Event falls man gewonnen hat
        else if (type == "won") {
            int sendClientId = _clientId.toInt();
            bool won = sendClientId == clientId;
            emit displayWonScreen(won);
        }
        // Event zum auswählen der Farbe von der Farbwechselkarte
        else if (type == "color") {
            emit pickColor();
        }
        // Event zum Anzeigen der Aktuellen Farbe von der Farbwechselkarte
        else if (type == "displayColor") {
            emit displayColor(response[1].at(0));
        }
        // Event zum Anzeige wie viel Karten gezogen werden müssen
        else if (type == "toDraw") {
            emit countDraw(response[1]);
        }
        // Event für chat Nachrichten
        else if (type == "message") {
            emit displayMessage((clientId == _clientId.toInt()), response[2]);
        }
    }
}
```

```
1
2  /*
3  * Legt die Karten im UI an für Spieler und Gegner
4  */
5  void MainWindow::drawCard(QString cardId, QString cardName) {
6      if(cardName != "back")
7      {
8          ClickableLabel *cardLabel = new ClickableLabel(Q_NULLPTR, Qt::WindowFlags(), cardId.toInt());
9          cardLabel->setObjectName(cardId);
10         cardLabel->setPixmap(QPixmap::fromImage(QImage(":/assets/" + cardName + ".png")));
11         cardLabel->setFixedSize(117, 171);
12         cardLabel->move(cardLabel->x(), 50);
13         cardLabel->setScaledContents(true);
14         cardLabel->setCursor(Qt::PointingHandCursor);
15         connect(cardLabel, &ClickableLabel::clicked, this, &MainWindow::onCardClick);
16
17         int spacing = -20; // Negative spacing for overlap
18         QSpacerItem *spacer = new QSpacerItem(0, 0, QSizePolicy::Fixed, QSizePolicy::Fixed);
19         spacer->changeSize(spacing, 0); // Adjust the size of the spacer
20
21         ui->primary_card_holder_layout->addWidget(cardLabel);
22         ui->primary_card_holder_layout->addSpacerItem(spacer);
23
24         if (ui->draw_count->text() == "" && this->isTurn) {
25             ui->skip_button->setVisible(true);
26         }
27     } else {
28         enemyCardCounter++;
29
30         if (enemyCardCounter <= 7) {
31             QLabel *enemyCardLabel = new QLabel();
32             enemyCardLabel->setPixmap(QPixmap::fromImage(QImage(":/assets/back.png")));
33             enemyCardLabel->setFixedSize(117, 171);
34             enemyCardLabel->move(enemyCardLabel->x(), 50);
35             enemyCardLabel->setScaledContents(true);
36
37             int spacing = -60; // Negative spacing for overlap
38             QSpacerItem *spacer = new QSpacerItem(0, 0, QSizePolicy::Fixed, QSizePolicy::Fixed);
39             spacer->changeSize(spacing, 0); // Adjust the size of the spacer
40
41             ui->enemy_card_holder_layout->addWidget(enemyCardLabel);
42             ui->enemy_card_holder_layout->addSpacerItem(spacer);
43         }
44
45         ui->enemy_card_counter->setText("Spieler 2 hat " + QString::number(enemyCardCounter) + " Karten");
46     }
47 }
```

```
1
2
3  /*
4  * Schickt Event play an den Server
5  * Wenn man auf eine Karte klickt
6  */
7  void MainWindow::onCardClick(int cardId, ClickableLabel *cardLabel) {
8      mainController->playCard(cardId);
9  }
```

```
1
2  /*
3  * Schick Event zum Server fürs Spielen einer Karte
4  */
5  void MainController::playCard(int cardId)
6  {
7      QString type = QString("play");
8      vector<QString> params = {QString::number(cardId)};
9      sendMessage(type, params);
10 }
```

4.2 Server-seitig

```
1
2  /*
3  * Liest und verabrebeitet Nachrichten von den Spielern
4  */
5  void Server::startRead(PlayerModel* player, PlayingFieldModel* pf){
6
7      // Dieser Slot wird aufgerufen, sobald der Client Daten an den Server sendet
8      // Der Server überprüft, ob es sich um einen GET-Request handelt und sendet ein sehr
9      // einfaches HTML-Dokument zurück
10
11     QTcpSocket *socket = player→getSocket();
12     qDebug() << "Data incoming";
13     qDebug() << socket→canReadLine();
14
15     QTextStream os(socket);
16     QStringList data = QString::fromLatin1(socket→readAll()).split(QRegExp(":"));
17     QString event = data[0];
18
19     qDebug() << data;
20     qDebug() << "Event: " << event;
21     qDebug() << "Requesting player id: " << player→getId();
22
23     // Event für erste verbinden
24     if(event == "connect")
25     {
26         os << "connected:" << player→getId();
27
28         socket→flush();
29
30         if(pf→hasSpace() == false)
31         {
32             pf→start();
33         }
34     }
35     // Event fürs Karte ziehen
36     else if(event == "draw")
37     {
38         pf→drawCard(player);
39     }
40     // Event fürs Spielen einer Karte
41     else if(event == "play")
42     {
43         pf→playCard(data[1], player);
44     }
45     // Event für überspringen des zuges
46     else if (event == "skip") {
47         pf→skip(player);
48     }
49     // Event für den Farbwechsel (Farbwechselkarte)
50     else if (event == "color") {
51         pf→pickColor(data[1], player);
52     }
53     // Event für Chat funktion
54     else if (event == "message") {
55         pf→chatMessage(data[1], player);
56     }
57
58     if ( socket→state() == QTcpSocket::UnconnectedState )
59     {
60         delete socket;
61     }
62 }
```

```
1
2  /*
3  * Spieler zieht eine Karte
4  * Überprüft ob spieler Karten ziehen darf und schickt event an spieler
5  */
6  void PlayingFieldModel::drawCard(PlayerModel *player)
7  {
8      if (this->currentPlayer != player || this->drawCardLimit == this->drawenCards) {
9          return;
10     }
11
12     CardModel* card = stack->getCard();
13     player->addCard(card);
14
15     logger->logToFile(player->getId(), "draw", card->getName());
16
17     for(PlayerModel* p : players)
18     {
19         QTcpSocket* pSocket = p->getSocket();
20         QTextStream os(pSocket);
21
22         if(player == p)
23         {
24             os << "card::" + card->getName() + ":" + QString::number(card->getId()) + "\n";
25         } else {
26             os << "card::back:0000\n";
27         }
28     }
29
30     this->drawenCards++;
31
32     if (this->drawCardLimit > 1) {
33         this->notifyDrawCards(player, this->drawCardLimit - this->drawenCards);
34     }
35
36     if (this->drawCardLimit == this->drawenCards) {
37         if (this->drawCardLimit > 1) {
38             this->switchPlayer(player);
39             this->drawenCards = 0;
40         }
41
42         this->drawCardLimit = 1;
43         this->notifyDrawCards(player, 0);
44     }
45 }
```

5 CSV-Ausschnitt

Player ID	Event	Data	Time
system	play	y3	08-04-2024 08-37-29
0	draw	go	08-04-2024 08-37-29
0	draw	r8	08-04-2024 08-37-29
0	draw	gu	08-04-2024 08-37-29
0	draw	b+	08-04-2024 08-37-29
0	draw	g4	08-04-2024 08-37-29
0	draw	g9	08-04-2024 08-37-29
0	draw	b1	08-04-2024 08-37-29
75	draw	r5	08-04-2024 08-37-29
75	draw	go	08-04-2024 08-37-29
75	draw	g+	08-04-2024 08-37-29
75	draw	y8	08-04-2024 08-37-29
75	draw	g2	08-04-2024 08-37-29
75	draw	g0	08-04-2024 08-37-29
75	draw	r2	08-04-2024 08-37-29
75	play	y8	08-04-2024 08-37-32
0	play	r8	08-04-2024 08-37-34
75	play	r5	08-04-2024 08-37-35
0	draw	r8	08-04-2024 08-37-39
0	play	r8	08-04-2024 08-37-39
75	play	r2	08-04-2024 08-37-41
…	…	…	…

6 Marketingstrategie

Unser Ziel ist es, ein herausragendes Uno-Multiplayer-Spiel zu entwickeln und erfolgreich zu vermarkten. Dieses Marketingkonzept dient dazu, eine umfassende Strategie festzulegen, um unser Spiel effektiv zu bewerben und eine breite Spielerbasis anzusprechen.

Zielgruppenanalyse

Die Hauptzielgruppen für unser Spiel umfassen Kinder, Jugendliche, Familien und Gaming-Enthusiasten. Durch eine sorgfältige Analyse der demografischen Merkmale, Interessen und Online-Verhaltensweisen dieser Zielgruppen können wir gezielte Marketingansätze entwickeln, um ihre Aufmerksamkeit zu gewinnen und ihr Interesse am Uno-Multiplayer-Spiel zu wecken.

Kinder:

Kinder sind eine wichtige Zielgruppe für unser Spiel, da Uno ein beliebtes Kartenspiel ist, das oft von Familien und in Schulen gespielt wird. Wir wollen die kinderfreundliche Natur des Spiels betonen und sicherstellen, dass unsere Marketingbotschaften altersgerecht und ansprechend sind. Dies kann durch bunte Grafiken, animierte Charaktere und spielerische Interaktionen erfolgen.

Jugendliche:

Jugendliche sind eine weitere wichtige Zielgruppe, da sie oft eine hohe Affinität zum Gaming haben und gerne Zeit mit Freunden online verbringen. Wir möchten die Wettbewerbsfähigkeit und soziale Komponente des Spiels betonen, indem wir Funktionen wie Ranglisten, Herausforderungen und Multiplayer-Modi hervorheben. Durch die Integration von Social-Media-Funktionen können Jugendliche ihr Spielerlebnis teilen und mit anderen Spielern interagieren.

Familien:

Familien stellen eine bedeutende Zielgruppe dar, da Uno ein Spiel ist, das generationsübergreifend gespielt werden kann und oft bei gemeinsamen Aktivitäten im Familienkreis verwendet wird. Wir möchten die gemeinschaftliche und

unterhaltsame Natur des Spiels betonen, indem wir Funktionen wie Familien-Turniere, Mehrspieler-Modi und spezielle Events für Eltern und Kinder anbieten. Durch gezielte Werbung in familienfreundlichen Medien und Veranstaltungen können wir Familien erreichen und ihr Interesse am Spiel wecken.

Gaming-Enthusiasten:

Gaming-Enthusiasten sind eine wichtige Zielgruppe, da sie oft nach neuen und innovativen Spielerfahrungen suchen. Wir möchten ihre Aufmerksamkeit durch die Integration von High-End-Grafiken, anpassbaren Einstellungen und kompetitiven Spielmodi gewinnen. Durch die Zusammenarbeit mit bekannten Gamern, Veranstaltung von Turnieren und Bereitstellung von exklusiven Inhalten können wir die Gaming-Community ansprechen und ihr Interesse am Uno-Multiplayer-Spiel wecken.

Hervorhebung des Einzigartigen Verkaufsvorschlags (USP)

Unser Uno-Multiplayer-Spiel zeichnet sich durch verschiedene einzigartige Merkmale aus, die es von anderen Uno-Implementierungen abheben. Dazu gehören eine herausragende Benutzerfreundlichkeit, zusätzliche Funktionen wie Turniere und eine ansprechende Grafik. Diese Aspekte werden in unserer Marketingkommunikation hervorgehoben, um potenzielle Spieler anzusprechen und von unserem Produkt zu überzeugen.

Benutzerfreundlichkeit:
Unser Spiel bietet eine intuitive Benutzeroberfläche und einfache Spielregeln, die es Spielern aller Altersgruppen ermöglichen, sofort einzusteigen und Spaß zu haben. Durch die Integration von Tutorials, Tipps und interaktiven Hilfeseiten möchten wir sicherstellen, dass neue Spieler sich schnell zurechtfinden und das Spiel genießen können.

Zusätzliche Funktionen:

Unser Spiel bietet eine Vielzahl von zusätzlichen Funktionen, die das Spielerlebnis bereichern und für Langzeitmotivation sorgen. Dazu gehören verschiedene Spielmodi wie klassisches Uno, Teamspiele, Spezialkarten und Herausforderungen. Durch regelmäßige Updates und die Integration von Spieler-Feedback möchten wir sicherstellen, dass unser Spiel immer wieder neue und spannende Inhalte bietet.

Ansprechende Grafik:

Die visuelle Präsentation unseres Spiels ist ein wichtiger Aspekt, der das Spielerlebnis verbessern und die Immersion fördern soll. Wir legen Wert auf hochwertige Grafiken, Animationen und Effekte, die das Thema des Spiels widerspiegeln und die Spieler in eine lebendige und unterhaltsame Spielwelt eintauchen lassen. Durch die Optimierung für verschiedene Bildschirmgrößen und Gerätetypen möchten wir sicherstellen, dass unser Spiel auf allen Plattformen gut aussieht und reibungslos läuft.

Aufbau einer Online-Präsenz und Social Media-Präsenz

Um eine starke Online-Präsenz aufzubauen und unsere Reichweite zu erhöhen, erstellen wir Profile auf den wichtigsten Social-Media-Plattformen wie Facebook, Twitter, Instagram und TikTok. Durch regelmäßiges Teilen von Updates, Spielanleitungen, Tipps und Tricks möchten wir die Community aktiv einbinden und ihr Interesse an unserem Spiel aufrechterhalten. Die Verwendung von relevanten Hashtags und gezielten Werbekampagnen ist entscheidend, um potenzielle Spieler anzusprechen und unsere Zielgruppen effektiv zu erreichen.

Community-Management:

Eine engagierte und aktive Community ist entscheidend für den langfristigen Erfolg unseres Spiels. Wir möchten eine offene und einladende Atmosphäre schaffen, in der Spieler ihre Gedanken, Ideen und Feedback teilen können. Durch die Moderation von Foren, Chats und Social-Media-Kanälen möchten wir eine positive und unterstützende Community aufbauen, die das Spielerlebnis verbessert und die Spielerbindung stärkt.

Influencer-Marketing und Partnerschaften

Durchführung von Influencer-Marketing:

Wir suchen aktiv nach Influencern in der Gaming-Community, die zu unserer Zielgruppe passen. Durch die Zusammenarbeit mit diesen Influencern möchten wir das Spiel einem breiteren Publikum vorstellen und ihr Interesse wecken. Dies kann durch die Erstellung von Let's Play-Videos, Rezensionen oder Sponsoring-Aktionen erfolgen, um die Glaubwürdigkeit unseres Spiels zu stärken.

Aufbau von Partnerschaften und Kooperationen:

Um unsere Reichweite weiter zu erhöhen, suchen wir nach Möglichkeiten für Partnerschaften mit anderen Unternehmen oder Organisationen, die eine ähnliche Zielgruppe ansprechen. Durch die Bereitstellung exklusiver Angebote oder Rabatte für deren Kunden oder Mitglieder streben wir eine win-win-Situation an, die nicht nur unsere Zusammenarbeit fördert, sondern auch neue Spieler für unser Spiel gewinnt. Durch die Identifizierung von potenziellen Partnern wie Gaming-Blogs, Veranstaltern von Spiele-Events oder Schulen können wir synergistische Beziehungen aufbauen und gemeinsame Marketingaktivitäten durchführen, um das Spiel einem breiteren Publikum vorzustellen. Diese Partnerschaften können auch dazu beitragen, das Vertrauen und die Glaubwürdigkeit unseres Spiels zu stärken, da sie von bereits etablierten und vertrauenswürdigen Quellen empfohlen werden.

Dieses Marketingkonzept bildet die Grundlage für unsere Marketingbemühungen und wird kontinuierlich überprüft und angepasst, um sicherzustellen, dass wir unsere Ziele effektiv erreichen.

7 Kostenplan

Stundensatz: 100,00 €
Stunden: 150h

Art	Kosten
Personalkosten	15.000,00 €
Sachmittelkosten	39,50 €
Softwarekosten	45,00 € für 2 Lizenzen Figma 658,00 € für 2 Lizenzen Qt for Application Development Professional
Hardwarekosten	Macbook Pro Leasing für 2 Entwickler: 43,03 €