Kurs: Oberstufe (4h)
Semester 1: Grundlagen der Programmierung mit C#

Thema	Fachwissen	Ziele für die Schülerinnen und Schüler
Logik mit Binäre Zahlen	• Wertigkeit der Ziffern im binären, dezimalen und hexadezimalen System • Umwandlung von Werten zwischen den Zahlensystemen. • Logische Operationen: UND, NICHT, ODER, EXCLUSIVES ODER • Optional: Logiksymbole in einfachen logischen Schaltplänen • Optional: Einfache Optimierung in der Auswertung logischer Ausdrücke. • Optional: Rechnen im binären System	• Können Zahlen ins binäre beziehungsweise ins dezimale Zahlensystem übertragen. • Können logische Schaltungen oder Logiktabellen vervollständigen. • Optional: Können einfache logische Optimierungen nach der Bool'schen Algebra vornehmen.
Algorithmen	• Definition • Kriterien: Determiniertheit, Determinismus, dynamische und statische Finitheit, Effektivität • Darstellung im Flussdiagramm oder als Blockdiagramm • Allgemein formulierte bedingte Abfragen • Allgemein formulierte Wiederholungen (Schleifen)	• Entwickeln Algorithmen für einfache alltägliche Prozesse. • Überprüfen Algorithmen und Arbeitsanweisungen auf ihre Kriterien hin: Umwandlung von Zahlen bezüglich unterschiedlicher Zahlensysteme, Intervallhalbierungsverfahren und Kochrezept. • Stellen Algorithmen grafisch (Flowchart) dar.
Compiler / Interpreter	• Grundlegende Unterschiede zwischen Compiler und Interpreter • Vor- und Nachteile der Konzepte	• Unterschiede, Vor- und Nachteile können benannt werden • Begründete problemabhängige Auswahl von Compilern oder Interpretern
Erste Programme C#	• Ausgabe von Informationen auf die Konsole. • Einlesen und Weiterverarbeiten von Informationen anhand von Tastatureingaben.	• Schreiben ein „Hallo-Welt-Programm“ • Lesen einfache Text-Information ein, modifizieren sie und geben diese wieder aus. „Hallo, Herr …, Sie sind … Jahre alt.““
Einfache Datentypen	• Variable als Platzhalter von Werten über einen Bezeichner ansprechbar. • Speicherbedarf, Operationen und Wertemenge von unterschiedlichen elementaren Datentypen. • Verarbeitungsgeschwindigkeit von ganzzahligen Typen im Vergleich zu den Gleitkommatypen. • Festlegung der möglichen Operationen, des Wertebereichs und des nötigen Speicherbereichs über die Zuordnung des Datentyps. • Variablendeklaration und -initialisierung bzw. -definition • Ganzzahlige Typen (byte, short, int, long) • Gleitkommazahlen (float, double, decimal) • Zeichenfolgen (string) und einzelne Zeichen (char). • Boolean-Typ. • Typenkonvertierung	• Unterscheiden von Variablendefinition und -deklaration. • Benutzen von Variablen in einfachen Programmen • Beispiel: Berechnung des Body-Mass-Index • Wählen den Variablentypen anhand vorgegebener Informationen (Wertebereich der Variablen, nötige Operationen), so dass der Speicherbedarf möglichst klein gehalten wird. • Unterscheidung zwischen der Ganzzahlarithmetik und der Gleitkommaarithmetik. • Sinnvolles Konvertieren von Datentypen unter Vermeidung von Informationsverlust.

Arithmetik	• Arithmetische Operatoren: +,*,-,/, +=, ++, … • Verkürzte arithmetische Operatoren +=, /=, %=, ... • Ganzzahlige Arithmetik (ganzzahlige Division, Modulo-Operator) • Reihenfolge der Auswertung von Operatoren • Logische Operatoren &&, ||, ...	• Beachtung der Operatorreihenfolge bei komplexen arithmetischen Ausdrücken. • Unterscheidung zwischen ganzzahliger und Gleitkommadivision. • Unterscheidungen der Auswertung bezüglich der Präfix- und der Postfixschreibweise.
Kontrollstrukturen	• Bedingte Abfragen (if-else, switch-case) • Schleifen (for, do-while, while) • break, contiune zur Steuerung von Schleifen • Vergleichende Operatoren ==, !=, >, >=, ...	• Schreiben von komplexeren Programmen: ◦ Binär-Dezimal-Konverter ◦ Mastermind Version 01 • Verwenden Schleifen bei Wiederholungen • Setzen Algorithmen in der Programmiersprache um • Erstellung einer Menüstruktur in einer Konsolenanwendung.
Arrays	• Felder (Arrays) – Definition • Strings als Arrays • Spezielle Stringfunktionen (trim, indexOf, ...) • foreach – Schleife im Vergleich zur for-Schleife • Felder dynamischer Länge	• Können Arrays erzeugen und sinnvoll benutzen • Können gezielt Einträge in Strings ändern bzw. auslesen. (Stringumkehrung). • Bespielprogramm: Schreiben eines Notenrechners mit Berechnung des Medians und des arithmetischen Mittelwertes. • Optional: Verbesserung von Mastermind
Methoden: Funktionen und Prozeduren	• Aufbau von Methoden (Funktionen und Prozeduren) ◦ Bezeichner ◦ Rückgabe-Datentyp ◦ Funktionsargumente • Übergabe von Argumenten anhand von Referenzen (ref, out) • Rekursion	• Können Funktionen zur Lösung von Teilproblemen programmieren. • Unterscheiden zwischen innerhalb von Funktionen unveränderlichen und veränderlichen übergebenen Variableninhalten. • Schreiben ein einfaches Programm zur Kryptographie nach dem Cäsar-Verfahren • Schreiben ein Programm zur Bestimmung von römischen Zahlen. • Können die Vorteile und die Nachteile von rekursiven Programmen erläutern. • Schreiben Mastermind 02 mit Funktionen als Konsolenanwendung.
Fehlerbehandlung	• Try, catch, throw • Ausgabe der intern generierten Fehlermeldung. • Ausgabe des Ausnahmetyps. • Unterschiedliche Behandlung von Ausnahmen nach Typ. • Generieren eigener Fehlermeldungen und Ausnahmetypen.	• Ergänzung bisheriger Programme um das Abfangen von Fehlern und gescheiterten Typenkonvertierungen. • Unterscheiden zwischen den verschiedenen Typen der möglichen Ausnahmen. • Können an sinnvollen Stellen innerhalb des Programms Ausnahmen auslösen.
Speichern	• Speichern aus dem eigenen Programm heraus. • Speichern von Variableninhalten als Text.	• Speichern und Einlesen von Informationen aus einem eigenen Programm heraus.

	• Speichern von Variableninhalten in Form im XML-Format. • Einlesen von Daten in das eigene Programm	
Umgang mit dem Visual Studio	• Erstellen und Verwalten von Programmprojekten. • Debugger: Setzen von Breakpoints, Überwachung von Variaben, Schreiben von programmgenerierten Debugmeldungen. • Erstellen und Verwenden von eigenen DLLs	• Sinnvolle Benutzung des Debuggers zur Beseitigung von Fehlern. • Ausgliedern von getesteten Funktionen in einer dynamische Bibliothek.

Semester 2: Objektorientiertes Programmieren und Computerspielentwicklung in C#

Vom Algorithmus zum Objekt	• Paradigmen der prozeduralen und der objektorientierte Programmierung. • Vor- und Nachteile der Paradigmen.	• Können den Unterschied in den grundlegenden Ansätzen der prozeduralen und der objektorientierten Programmiersprachen benennen. • Wählen anhand einer Problemstellung einen geeigneten Ansatz aus.
OOP	• Klasse als Objektbeschreibung (Bauplan) • Verschiedene Objekte (Stift, Vierbeiner, Auto) • Methoden (Was kann …?) und Attribute (Wie ist. ..?) • Gemeinsamkeiten unterschiedlicher Klassen → Vererbung • Objektorientierte Modellierung (mit UML) Klassendiagramme • Vererbung von Rechten: private, public, protected • Prinzip der Datenkapselung.	• Zeichnen Klassendiagramme mit UML für einfache Probleme • Erweitern Klassendiagramme mit UML • Implementierung eines Modells • Sinnvolles Setzen von Zugriffsrechten.
Schreiben von Klassen	• Erstellung einer Klasse • Constuctor, Destructor • Attribute, Methoden • private, protected, public, internal • Klassenvariablen (static) und Objektvariablen • Überladung von Operatoren • Erzeugen von Objekten (Instanzen) einer Klasse. • Statische Klassen	• Können anhand einer Modellierung eine Klasse erstellen und deren Objekte erzeugen, benutzen und vernichten. • Beispiele: Klassen von geometrischen Objekten, Photon und Klasse Vektor → 3DVektor
Vererbung	• Vererben von Klassen • Überschreiben von Methoden • Mehrfaches Vererben • Interfaces	• Können eine Elternklasse vererben und anpassen. • Können Interfaces sinnvoll in verteilten Programmprojekten einsetzen.
Einfachste Windowsprogrammierung als Anwendung der OOP	• Programmieren von Dialogen über DotNet • Erzeugung von Fenstern / Forms • Setzen von Eigenschaften • Abfragen /Abfangen von Ereignissen • Einlesen von Werten über Textelemente innerhalb eines Fensters	• Können einfachste Windowsanwendungen erzeugen. • Beispielprogramm: Taschenrechner mit einfacher Texteingabe, oder einer kleinen graphischen Anwendung.
Spielprojekt	• Entwicklung von Computerspielen als größere Programmieraufgabe von der Idee zum Produkt. • Eigenständige Entwicklung eines Computerspiels in Kleingruppen von maximal 4 Personen. • Aufgabenaufteilung in einem Team	• Eigenständige Entwicklung einer Spielidee. • Anwendung der Struktur: Ideenfindung, Planungsphase, Designkonzept, Entwicklungsstechnische Umsetzung, Testphasen

	• Umsetzung dieser Idee anhand der Programmierung • Testen der Spielmechanik • Gegebenenfalls Programmierung einer Spielintelligenz (AI). • Gegebenenfalls selbstständiges Einarbeiten in einer Grafikbibliothek.	• Präsentation der Überlegungen zum Projekt einschließlich eines Zeitplans und der geplanten Aufgabenaufteilung. • Präsentation von regelmäßigen Statusberichten einschließlich der Reflexion hinsichtlich der Zeit- und der Projektplanung. • Abschlusspräsentation mit Reflexion bezüglich der erreichten und der nicht erreichten Meilensteine. • Arbeiten an einem Projekt mit verteilten Verantwortlichkeiten. • Arbeiten mit einer professionellen Spielentwicklungssoftware Unity oder Unreal.