Computer Science with Game Development

Course Catalogue

Programme overview:

The Computer Science Program with Game Development is a 2.5-year program on bachelor level. The English-taught exchange semester (4th semester) is only offered in the spring. Exchange students will join our Danish and other exchange students.

Modules	Component Code	ECTS
Game AI	3S-IM	10
Development with JavaScript and React	3S-EC	10
Machine Learning	3S-MC	10
Development in a Modern Game Engine	3S-BL	10
Development with Python	3S-SM	10
Unreal & C++	3S-DA	10

Game AI

Autonomous Agents and Decision Making in Games

Build game agents that react, remember, plan and cooperate. In this practical module, you develop increasingly sophisticated AI behaviours in Unity and C#, progressing from simple rules to behaviour trees and goal-oriented planning. A team-based arena game provides a shared testbed for exploring how individual decisions shape collective performance.

Module at a glance

Institution Dania Games, Dania Academy, Grenaa/Silkeborg, Denmark

Credits and duration 10 ECTS • approximately 275 hours of total study • six-week teaching and project sequence

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development: object-oriented C#, software design patterns and UML.

Technical environment Unity, C# and the supplied Beast ball game framework, with example agents and supporting documentation.

What you will explore

How does an agent make useful decisions with incomplete information? When should it respond immediately, rely on memory or construct a plan? How can a team coordinate when its members have different roles and local knowledge? The module connects these questions to concrete software architectures and observable behaviour in a running game.

The core focus is agent-based game AI: perception, internal world models, decision making, planning and coordination. Movement, decision making and strategy provide the organising framework. Software engineering is integrated throughout, with attention to modularity, extensibility, debugging and performance.

Curriculum and progression

1 Agent foundations and reactive behaviour Analyse game environments, percepts and observation histories; distinguish objects from agents and examine observability, uncertainty and change. Explore deduction, induction and abduction, then implement condition–action rules and an agent factory.

2 Finite state machines Model states, transitions and triggers using state diagrams. Implement behaviour with the State pattern, compare State and Strategy, and examine the limits of increasingly complex state machines.

3 Model-based agents Maintain an internal world model from observations over time. Represent known enemies and power-ups, reason about incomplete information and manage knowledge that becomes outdated.

Curriculum and Learning Outcomes

4 Behaviour trees and shared knowledge Construct selector, sequence, decorator and leaf nodes. Use blackboards to distinguish personal and shared information and support reusable behaviour structures.

5 Goals and action planning Compare rule-based, priority-based and utility-based goal selection. Work with Goal-Oriented Action Planning (GOAP), including world states, action preconditions, effects and costs; examine the computational trade-offs of planning.

6 Team strategy and evaluation Develop roles and coordination through shared knowledge. Test, tune and compare agents in competitive matches, culminating in a five-versus-five tournament. Utility AI, pathfinding and steering provide related topics for further exploration.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain and compare reactive, state-based, model-based, behaviour-tree and goal-based agent architectures, including their assumptions, limitations and computational trade-offs.
- Analyse a game environment and define what an agent can perceive, remember and act upon under conditions of partial observability and uncertainty.
- Implement reactive rules, finite state machines and internal world models; construct behaviour trees with blackboards; and implement goal selection and a GOAP planner.
- Apply Abstract Factory, State and Strategy patterns to structure AI code, and document designs using UML class diagrams, state diagrams and behaviour-tree diagrams.
- Select, justify and combine AI techniques for a game or simulation problem, and evaluate behaviour, extensibility, maintainability and performance through testing and debugging.
- Contribute to collaborative AI development, communicate design decisions and independently investigate further game AI techniques.

Learning through development

Short theory sessions lead directly into practical implementation. Students iteratively extend an agent, test it against supplied examples and peers, and use the results to refine their designs. Guided exercises, independent study and project supervision support a final individual or group project. The tournament is a compulsory learning activity governed by shared rules and an honour code; it provides evidence for analysis and reflection.

Project and assessment

The project produces a tournament-ready agent and factory, source code, design documentation and a video walkthrough of up to five minutes. Students explain their architecture, demonstrate behaviour and reflect on testing and tournament experience.

In the programme, assessment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.

Development with JavaScript and React

Interactive Web Applications and Browser Games

Create interactive web applications using JavaScript and React. This practical module connects HTML and CSS with asynchronous programming, reusable components and state-driven interfaces. Games and playful interactions provide examples and possible project contexts alongside broader web applications, allowing you to explore how user input, application data and interface updates work together.

Module at a glance

Institution Dania Games, Dania Academy, Silkeborg/Grenaa, Denmark

Credits and workload 10 ECTS • approximately 275 hours of total study

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development, including object-oriented programming and software design. JavaScript and React are introduced within the module, drawing on comparisons with C#.

Language English

Technical environment HTML, CSS, JavaScript, React, JSX, browser APIs, browser debugging tools and React DevTools.

What you will explore

How do you turn a static page into an application that responds to users and changing data? How should state be organised across components? What happens while an application waits for data? The module connects web fundamentals to the architecture and behaviour of interactive React applications.

Game examples can make state changes and user feedback tangible: a quiz, turn-based game or score-tracking interface can connect programming concepts to visible results. The wider focus remains web application development, including component design, data handling, debugging and integration with browser technologies.

Curriculum and progression

1 Web foundations Use HTML to structure content and CSS to control presentation. Understand how these technologies work with JavaScript to create interactive web interfaces.

2 JavaScript programming Work with types, variables, objects and functions. Compare JavaScript's dynamic typing and type coercion with familiar C# behaviour, and use language features to manipulate application data.

3 Asynchronous behaviour and browser APIs Use callbacks and Promises to organise asynchronous work. Retrieve and manipulate data through browser APIs, and connect external data to an application's interface.

Curriculum and Learning Outcomes

4 React and component architecture Build interfaces using JSX, function components and class components. Explore props, component composition, the virtual DOM and the relationship between component updates and rendered output.

5 State input and lifecycle Implement controlled components and handle user input. Use Hooks and explore class component lifecycle methods. Apply spread syntax and arrays of components to construct interfaces from changing data.

6 Integration and application development Combine components, state, styling and data access in a coherent application. Use browser debugging tools and React DevTools to investigate behaviour and performance, progressing from practical exercises to project and examination-project work.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain JavaScript's role in web development, its typing behaviour and asynchronous model, and its relationship with HTML, CSS and browser APIs.
- Implement JavaScript functionality using objects, functions, callbacks and Promises, and retrieve and manipulate data through browser APIs.
- Structure React applications using JSX, function and class components, props and component composition, explaining how interface updates relate to the virtual DOM.
- Manage state and user input using controlled components, Hooks and relevant lifecycle methods, and render collections of components from application data.
- Test and debug JavaScript and React applications using browser tools and React DevTools, and investigate interface behaviour and performance.
- Integrate React with other web technologies to develop an interactive application, justify design choices and extend existing programming skills to a new language and library.

Learning through development

The module progresses from HTML, CSS and JavaScript foundations to React components and application development. Practical exercises connect individual concepts to visible interface behaviour, followed by project work and an examination project.

Games are used as example material and may also form the submitted product. Suitable contexts include a browser-based quiz, a turn-based game or an interactive game companion. Other web applications can demonstrate the same competencies in state management, user interaction and component-based design.

Project and assessment

The examination project provides a context for integrating JavaScript, React, HTML and CSS into a working web application. A game is one possible product. The application should demonstrate coherent component structure, state and input handling, and appropriate integration with browser functionality.

Assessment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.

Machine Learning

Learning from Data with Python and Neural Networks

Explore how software can learn from examples and use that knowledge to solve practical problems. This applied module introduces machine learning through visual experiments, Python programming and work with existing tools and models. You move from understanding the learning process to preparing data, training models and evaluating whether their predictions are useful.

Module at a glance

Institution Dania Games, Dania Academy, Skive/Hybrid, Denmark

Credits and workload 10 ECTS • approximately 275 hours of total study

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development. No previous machine learning experience is required; Python and numerical data handling are introduced in the module.

Language English

Technical environment Python, NumPy, browser-based neural network demonstrations, online notebook environments, and applied tools and model resources including Roboflow and Hugging Face.

What you will explore

What does it mean for a machine to learn? How does a neural network adjust its behaviour from examples, and how can you tell whether it has learned something useful? The module connects these questions to classification, regression and the practical decisions involved in building a machine learning solution.

The central emphasis is supervised learning and deep neural networks. Convolutional and recurrent neural networks and reinforcement learning introduce a broader range of approaches. Applications may include image recognition and prediction, with scope to investigate game-related data or interactive applications where these fit the problem and available data.

Curriculum and progression

1 Machine learning foundations Distinguish artificial intelligence, machine learning and deep learning. Explore supervised learning, classification and regression, and relate learning methods to the kinds of problems they can address.

2 Neural networks and the learning process Use visual tools to investigate inputs, outputs, hidden layers and learned representations. Develop a conceptual understanding of prediction errors, parameter updates and backpropagation through small experiments.

3 Python and data preparation Move from visual experiments to Python implementations. Use NumPy to manipulate numerical data, collect and organise examples, and prepare inputs and target values for training and testing.

Curriculum and Learning Outcomes

4 Training and evaluating models Train and test models for practical classification and regression tasks. Compare predictions with expected results, investigate errors and examine whether a model performs usefully on data beyond its training examples.

5 Architectures and learning approaches Build introductory knowledge of convolutional neural networks, recurrent neural networks and reinforcement learning. Compare their purposes and the roles of data, feedback and model structure. Genetic algorithms may be explored as an optional extension.

6 Applied tools and model integration Explore workflows using Roboflow and Hugging Face alongside Python-based development. Select relevant tools or frameworks, structure machine learning components and investigate how a trained or existing model can support an application.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain how machines learn from data, including the principles of neural networks, training and backpropagation at a conceptual level.
- Distinguish classification and regression and describe the purposes of convolutional neural networks, recurrent neural networks and reinforcement learning.
- Collect, prepare and manipulate data for machine learning using Python and numerical data tools.
- Implement and structure machine learning components, and use an appropriate framework or tool to train and test a model for a practical task.
- Test, debug and evaluate a machine learning solution, interpret prediction errors and discuss the relationship between data, training and observed performance.
- Select and communicate a suitable machine learning approach to a practical problem, explaining the model's behaviour, limitations and potential application.

Learning through development

Visual demonstrations provide an accessible entry point before students reproduce and extend experiments in Python. Practical work connects the learning process to data preparation, model training and testing, supported by introductory theory and exploration of applied tools.

The proposed project approach centres on a manageable classification or regression problem. Students investigate data, develop a working solution and use evidence from testing to explain their results. A game-related application is one possible context; the module also supports broader software and data applications.

Project and assessment

A proposed final project demonstrates a practical machine learning workflow: problem definition, data preparation, model training or adaptation, testing and interpretation of results. The working solution is supported by an explanation of the chosen method, relevant code and evidence of model performance.

Assessment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.

Development in a Modern Game Engine

Interactive Applications with Unity and C#

Build interactive applications by bringing code, digital assets and engine systems together. In this practical module, you use Unity and C# to develop games, simulations or interactive visualisations. You explore the complete development workflow, from an initial prototype to a distributable application, and learn how a game engine supports both creative experimentation and structured software development.

Module at a glance

Institution Dania Games, Dania Academy, Grenaa, Denmark

Credits and workload 10 ECTS • approximately 275 hours of total study

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development: object-oriented C#, software design principles and design patterns. Unity is introduced as a development environment within the module.

Language English

Technical environment Unity, C#, the Unity Editor, integrated engine systems, digital assets, debugging and profiling tools, and build workflows.

What you will explore

How does a scene become an interactive application? How do scripts, components and assets work together at runtime? How can you organise a project so that it remains understandable as it grows? The module makes the engine workflow accessible while building on your existing programming and software design skills.

Unity provides a shared environment for exploring gameplay, simulation and visual communication. The emphasis is on using the engine effectively: composing behaviour, integrating content, diagnosing problems and preparing an application for distribution. You connect technical decisions to the needs of the intended user and the purpose of the application.

Curriculum and progression

1 The editor and runtime Understand Unity's structure and workflow. Create and configure projects, organise scenes, and explore the relationship between GameObjects, components, prefabs and assets in the editor and at runtime.

2 C# and component-based development Implement interactive functionality by combining C# scripts with components, events and engine systems. Relate Unity's execution model to application behaviour and apply object-oriented principles within a component-based architecture.

3 Interaction and presentation Use relevant systems for user interaction, physics, animation, sound and visual presentation. Combine these systems into a coherent experience for a game, simulation or interactive visualisation.

Curriculum and Learning Outcomes

4 Assets and the content pipeline Import, configure and integrate digital assets. Organise content, scenes and prefabs to support reuse, consistent behaviour and maintainability, and understand how code and digital content come together in a Unity project.

5 Project architecture and collaboration Structure scripts and assets around clear responsibilities. Apply suitable design patterns and reusable components, and establish project organisation that supports collaboration on code and content.

6 Debugging and delivery Test functionality, diagnose problems and use profiling to investigate performance. Refine the application, configure a build and prepare a distributable version, connecting development decisions to observed results.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain the principles and working methods of interactive real-time development, including Unity's editor, runtime, execution model and principal engine systems.
- Configure Unity projects and develop interactive functionality by combining C# scripts, components, events and systems for interaction, physics, animation, sound and visual presentation.
- Organise scenes, scripts, prefabs and assets for reuse and maintainability, applying object-oriented principles, separation of responsibilities and appropriate design patterns.
- Import, configure and integrate digital assets within a coherent content pipeline, and combine programming, engine functionality and content in a working application.
- Test and debug functionality, use profiling tools to assess performance, and select, implement and communicate solutions to practical development problems.
- Manage the development of a small Unity application from prototype to release, contribute to professional and interdisciplinary collaboration, and acquire new knowledge of engine tools, packages and development methods.

Learning through development

Practical assignments build towards a group or individual development project. Students apply existing programming knowledge in Unity, explore how engine systems interact and use testing and debugging to explain the behaviour of their applications.

A small application provides a context for connecting code, content and user interaction. The application may take the form of a game, simulation or interactive visualisation. Development work emphasises clear project structure, reusable components and the ability to explain technical choices and their consequences.

Project and assessment

The final project takes a Unity application from prototype to a distributable build. It brings together C# programming, engine systems, digital assets and user interaction, supported by documentation of the application structure, development choices and relevant functional and performance checks.

In the programme, assessment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.

Development with Python

Applications, Games and Concurrent Programming

Use your existing programming experience to develop applications and games in Python. This module combines language foundations with object-oriented design, concurrent programming, network communication and practical game development. You explore how Python differs from C# and learn to choose language features, libraries and execution models that suit the problem you are solving.

Module at a glance

Institution Dania Games, Dania Academy, Grenaa, Denmark

Credits and workload 10 ECTS • approximately 275 hours of total study

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development, including object-oriented C#, software design patterns and modelling. Python is introduced as a new programming language.

Language English

Technical environment Python, standard libraries for threading, multiprocessing and network communication, and Pygame for interactive game development.

What you will explore

How do you transfer your C# knowledge to a dynamically typed language? What changes when work is divided between threads or processes? How can programs exchange information across a network? The module connects language design and runtime behaviour to concrete applications.

Games provide examples of input handling, animation, collision and communication, alongside conventional software applications and scripting. You work with both everyday Python techniques and more advanced language mechanisms, considering their effect on clarity, abstraction and program quality.

Curriculum and progression

1 From C# to Python Compare syntax, typing and common programming idioms. Work with variables, loops, collections and scripts, and transfer existing knowledge of algorithms and program structure to Python.

2 Objects and language mechanisms Develop functions, methods, classes and constructors. Explore duck typing, polymorphic behaviour, decorators and metaclasses, relating these mechanisms to abstraction, reuse and the readability of a design.

3 Threads and coordination Distinguish concurrency from parallel execution. Work with threads, locks, semaphores and condition variables; use context managers and the with statement for resource handling. Examine the Global Interpreter Lock and its implications in the interpreter configuration used.

Curriculum and Learning Outcomes

4 Processes and shared work Use multiprocessing to organise work across processes. Apply the main-module guard and explore interprocess communication using queues and pipes, together with shared values, arrays and managers.

5 Network communication Implement TCP and UDP clients and servers. Compare communication patterns and explore unicast, multicast and broadcast, using small networked applications or game-related examples to examine message exchange.

6 Games with Pygame Set up Pygame and build an interactive application with sprites, keyboard input, animation and collision handling. Combine language features and libraries in a small game or conventional software project.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain Python's typing and abstraction mechanisms, compare them with C# and relate implementation choices to program quality and maintainability.
- Implement applications using functions, classes, collections, data structures and appropriate design patterns, drawing on existing object-oriented programming knowledge.
- Use Python libraries and language mechanisms to structure reusable software, and document designs according to professional conventions.
- Develop concurrent and multiprocessing solutions using appropriate coordination and communication mechanisms, and explain relevant execution constraints.
- Implement basic TCP and UDP communication and develop interactive functionality, including game input, sprites, animation and collision through Pygame.
- Contribute professionally to a software development project, communicate design choices and independently acquire further Python techniques and tool knowledge.

Learning through development

Language exercises connect familiar C# concepts to Python practice before extending into concurrency, networking and game development. Students investigate how design decisions affect program behaviour and use small implementations to build understanding of more advanced mechanisms.

Game examples provide a practical setting for combining input, state, presentation and program logic. A proposed project may be a game or a conventional software application, with its scope chosen to demonstrate relevant programming and software design skills.

Project and assessment

A proposed final project brings Python language features and appropriate libraries together in a working application. Supporting documentation explains the software structure, design choices and relevant checks of functionality. Concurrency or networking can be included where appropriate to the project.

Assesment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.

Unreal & C++

Game Development with C++ and Blueprints

Turn game ideas into playable experiences using Unreal Engine, C++ and visual programming. In this practical module, you combine Blueprint scripting with C++ development to build gameplay systems, interactive levels and user interfaces. You explore how a commercial game engine organises a game and learn to make informed decisions about architecture, implementation and code quality.

Module at a glance

Institution Dania Games, Dania Academy, Grenaa, Denmark

Credits and workload 10 ECTS • approximately 275 hours of total study

Placement Fourth-semester elective in the Danish AP Degree programme in Computer Science (Datamatiker). Offered for consideration as a second-year undergraduate exchange module, subject to the partner institution's academic review.

Entry requirements Prior study equivalent to introductory and intermediate programming and systems development: object-oriented programming, software design patterns and design modelling. The module introduces C++ in the context of Unreal Engine.

Language English

Technical environment Unreal Engine 5, C++, the Blueprint visual scripting system, editor tools and UI widgets.

What you will explore

How do you translate a game mechanic into engine components and code? Which responsibilities belong in an Actor, a Controller or a Character? When is a Blueprint the right tool, and when does a C++ implementation provide a more suitable foundation? The module connects these decisions to practical game development and software design.

The central focus is the relationship between engine architecture, programming and the player experience. You work across code, visual scripting, level construction and interface design, using abstraction and design patterns to create understandable, reusable and maintainable systems.

Curriculum and progression

1 Unreal Engine and the editor Understand Unreal Engine as a software system and game development platform. Navigate the main editor, level and scene editing tools, and node-based editors; assemble and organise the elements of an interactive game environment.

2 C++ within Unreal Engine Use C++ to implement game functionality through the Unreal Engine API. Work with engine-specific classes and data types, including strings and collections, and apply object-oriented design to the structure of game code.

3 Gameplay classes and components Create and extend Actors, Components, Pawns, Characters and Controllers. Allocate responsibilities across these classes and use component-based composition to structure gameplay behaviour.

Curriculum and Learning Outcomes

4 Blueprint visual programming Create and edit Blueprint graphs and level Blueprints. Understand Blueprint compilation and execution, implement game logic in the node editor and debug visual scripts.

5 Connecting C++ and Blueprints Combine C++ implementations with Blueprint functionality. Use events and delegates to connect behaviours, and justify the division of responsibilities between code and visual scripting.

6 Playable experiences and software quality Bring gameplay, level design and UI widgets together in a game project. Apply design patterns, debug behaviour across C++ and Blueprints, and document the resulting software design.

Learning outcomes

On successful completion of the module, students will be able to:

- Explain Unreal Engine's role as a game development platform, the responsibilities of its principal gameplay classes and the relationship between C++ and Blueprint execution.
- Use C++ and the Unreal Engine API to implement gameplay functionality, apply abstraction mechanisms and work with engine-specific data types, components and libraries.
- Develop and debug Blueprint scripts, connect them to C++ systems and justify implementation choices between visual scripting and code.
- Model and implement game systems using appropriate design patterns, including Component, Decorator and Facade, and assess their contribution to software quality.
- Create interactive levels and user interfaces using Unreal Engine's editor tools, Blueprints and widgets, relating implementation decisions to gameplay and usability.
- Document software designs in line with professional conventions, contribute to a collaborative development project and independently acquire further skills in C++, engine tools and game programming.

Learning through development

Practical assignments build towards a group game development project. Students work with both code and visual scripting, using the editor to connect individual systems into a playable product. Programming, software modelling and game design are treated as connected activities: implementation choices are examined in relation to functionality, maintainability and the intended player experience.

Mandatory assignments provide milestones before the final project. They are assessed as approved or not approved, and all must be submitted and approved for the student to be eligible for the examination.

Project and assessment

Students work in groups in response to a project brief issued at the start of the project period. The brief specifies the required product and submission requirements. The project provides an opportunity to demonstrate the integration of Unreal Engine tools, C++, Blueprints and software design in a coherent game development solution.

In the home programme, assessment is a joint individual oral examination covering three electives, supported by a shared synopsis of 3–6 standard pages. The 40-minute examination includes presentation, discussion and deliberation. Internal assessment uses the Danish seven-point grading scale, with one combined grade based on the learning outcomes.