Liverpool John Moores University

Warning: An incomplete or missing proforma may have resulted from system verification processing

Title:	PROGRAMMING FOR GAMES
Status:	Definitive
Code:	**4107COMP** (121205)
Version Start Date:	01-08-2021
Owning School/Faculty:	Computer Science and Mathematics
Teaching School/Faculty:	Computer Science and Mathematics

Team	Leader
Silvester Czanner	Y
Abdennour El-Rhalibi	

Academic Level:	FHEQ4	**Credit Value:**	20	**Total Delivered Hours:**	55
Total Learning Hours:	200	**Private Study:**	145		

Delivery Options
Course typically offered: Semester 1

Component	**Contact Hours**
Workshop	55

Grading Basis: 40 %

Assessment Details

Category	**Short Description**	**Description**	**Weighting (%)**	**Exam Duration**
Test	AS1	In class test	40	
Artefacts	AS2	Design, develop and test an application to specification.	60	

Aims

To gain experience with the tools which form the ecosystem used to develop, compile, debug and test code using an appropriate high-level programming language.
To develop problem solving and programming skills to enable the student to design solutions to non-trivial problems and implement those solutions in a high-level

language.
To relate software engineering and fundamental programming skills to computer games development.
To build a foundation for more advanced programming techniques, including object-oriented design and programming and the use of standard data structures and algorithms.

Learning Outcomes

After completing the module the student should be able to:

1	To build simple data models via primitive types and arithmetically/logical manipulate data to solve a specific simple programming task.
2	Utilise the fundamental language-level control structures to control program flow.
3	Evaluate and select appropriate data structures and algorithms from the standard libraries of a high-level language and apply them in order to solving common programming problems.
4	Use software engineering design techniques to decompose an application specification into a set of data models and algorithmic operations.
5	Compose user-defined structures and functions to model real-world data and simple algorithmic processes.
6	Apply the Software Development Lifecycle to build a program to specification using an appropriate high-level language.

Learning Outcomes of Assessments

The assessment item list is assessed via the learning outcomes listed:

In class test	1	2	3			
Design, Development and Test	1	2	3	4	5	6

Outline Syllabus

Type Theory:
- Bits, bytes, and words. Numeric data representation and number bases.
- Representation of non-numeric data (character codes, graphical data)
- Variables and primitive data types (e.g., numbers, characters, Booleans)
- Compound types built from other types (Strings, static and dynamic arrays, functions, references, Heterogeneous aggregates)
- A type as a set of values together with a set of operations
- Association of types to variables, arguments, results, and fields
- Type safety and errors caused by using values inconsistently given their intended types
- Goals and limitations of static typing

Language and Program-Level Constructs:
- Basic syntax and semantics of a higher-level language. Syntax vs. semantics
- Expressions and assignments.

- Propositional logic / Logical connectives / Truth tables
- Simple I/O including file I/O
- Conditional and iterative control structures
- Functions and parameter passing
- References and aliasing.
- Strategies for choosing the appropriate data structure.
- Simple numerical algorithms, such as computing the average of a collection of numbers, finding the min, max, and mode in a collection.
- Programming using library components and language-level APIs

Software Development Fundamentals:
- The concept of a specification
- System design principles: separation of concerns, information hiding, coupling and cohesion, re-use of standard structures
- Testing fundamentals and test-case generation
- Simple refactoring
- Modern programming environments
- Debugging strategies
- Program comprehension
- Program correctness
- Types of errors (syntax, logic, run-time)
- Eliminating some classes of errors without running the program
- Documentation, Comments and program style
- Structured design (top-down functional decomposition), object-oriented analysis and design.

Architectural Principles Related to Programming:
- Interpretation vs. compilation to native code vs. compilation to portable intermediate representation
- Language translation pipeline: parsing, type-checking, translation, linking, execution
- Execution as native code or within a virtual machine and dynamic (or "just-in-time") code generation
- Run-time layout of memory: call-stack, heap, static data
- Manual memory management: allocating, de-allocating, and reusing heap memory.
- Automated memory management: garbage collection as an automated technique using the notion of reachability.

Learning Activities

Workshop – Tutor-led practical session in the computer laboratory to cover both theories and techniques of programming

Further exercises – additional exercises for students to work on in their own time.

Directed learning – provides additional reading to enable practical work to be completed.

Learning materials can be accessed digitally via University Virtual Learning Environment (VLE).

Notes

In this module, students will develop their high-level programming skills using the industry standard languages for computer games development. Students will be introduced to the fundamental concepts of data type creation, utilisation and specification, programmatic computation, logic and how to control application flow. Practical experience with a high-level language and its associated ecosystem will lead to the development of problem solving and decomposition skills in order to solve real-world development problems using the processes of the software development lifecycle.