Games Engineering II – CW208

Module Information (CW208)

- **Module Title:** Games Engineering II
- **Credits:** 10
- **Level:** 8 (Final year undergraduate)
- **Languages Used:** C++11/14, Erlang, Haskell

Learning Outcomes

By completing this module, students will be able to: 1. Select, design, and implement testing strategies appropriate to games development.
2. Evaluate methods for synchronising concurrent processes and their effects in game execution environments.
3. Design algorithms that execute on multiple cores or processors.
4. Develop functional programs and integrate them into digital games.
5. Apply suitable design patterns in games.

Indicative Content

- Testing: unit, automated, coverage techniques, test management.
- Concurrent programming: threads, multicore, GPU programming, APIs (OpenMP, MPI, CUDA).
- Game programming patterns: case studies and design patterns.
- Game engine architecture: implementation techniques.
- Functional programming: process creation, message passing, distributed logic.

Assessment Breakdown

- Continuous Assessment: **20%**
- Project: **20%**
- Practical labs: **20%**
- Final exam: **40%**

🕮 Recommended Books

1. *Modern C++ Programming with Test-Driven Development* – Jeff Langr
2. *Game Programming Patterns* – Robert Nystrom
3. *Game Engine Architecture* – Jason Gregory
4. *C++ Concurrency in Action* – Anthony Williams
5. *Parallel and Concurrent Programming in Haskell* – Simon Marlow
6. *Programming Erlang* – Joe Armstrong

Get GIT Access

https://forms.office.com/e/47djqVwMEf

🗁 Repository Structure

```
GamesENGII/
├── LabNotes/          # Instructor-provided lab handouts (weekly)
├── LabCodes/          # Instructor reference/starter code (weekly)
├── studentLabs/       # Each student has their own folder here
│   └── SETU202212/    # Example student folder with registration SETU202212
└── .github/
    └── CODEOWNERS
```

- **LabNotes/** → Posted by instructor weekly.
- **LabCodes/** → Reference/starter codes for each Lab. Solutions will be updated at the end of each week.
- **studentLabs/** → Each student must work *only* in their own folder, named exactly as their registration code.

🚀 Student Setup (Option A – HTTPS)

1. Clone the repository

```
git clone https://github.com/oa-SETU/GamesENGII.git
cd GamesENGII
```

2. Configure Git identity (one-time)

```
git config --global user.name  "Your Name"
git config --global user.email "your.email@example.com"
```

3. Create your personal folder (first lab)

Your folder must be named exactly as your registration code:

```
REG="SETU202212"   # This is the example code, replace with your own registration code
mkdir -p "studentLabs/$REG"
```

This folder will hold all your weekly lab submissions.

👨‍💻 Weekly Workflow

Each lab week, follow this cycle:

1. Sync with main

Always pull the latest course material before starting:

```
git checkout main
git pull --rebase
```

2. Create a branch for this lab

Use the naming format: labNN/<REG>
Example for Week 1:

```
LAB="lab01"
git checkout -b "${LAB}/${REG}"
```

3. Create your lab folder & starter files

```
mkdir -p "studentLabs/$REG/${LAB}"

# Example C++ starter file
cat > "studentLabs/$REG/${LAB}/main.cpp" << 'CPP'
#include <iostream>
int main() {
    std::cout << "Hello, GamesENGII!\n";
    return 0;
}
CPP
```

```
# Or Add your files directly from your IDE

# Optional README for the lab
cat > "studentLabs/$REG/${LAB}/README.md" << 'MD'
# Lab01 Submission
- Name: ...
- Registration: ...
- How to build: g++ -std=c++17 main.cpp -o app && ./app
MD
```

4. Stage and commit

```
git add "studentLabs/$REG/${LAB}"
git status    # verify only your folder is staged
git commit -m "${LAB}: submission for ${REG}"
```

5. Push your branch

```
git push -u origin "${LAB}/${REG}"
```

6. Open a Pull Request

- Go to the repo on GitHub.
- Click **"Compare & pull request."**
- Title format: lab01 – SETU202212.
- Ensure only files under your folder are included.
- Submit. Instructor reviews & merges.

Updating a Submission

If you need to revise work for an open PR:

```
# edit your files
git add "studentLabs/$REG/${LAB}"
git commit -m "${LAB}: fix and update"
git push
```

The same PR updates automatically.

Next Week

Start from main and repeat:

```
git checkout main
git pull --rebase
LAB="lab02"
git checkout -b "${LAB}/${REG}"
```

⚠ Rules & Restrictions

- Work **only inside** studentLabs/<YOUR_REG>/.
- Do **not** edit or delete files in other folders.
- Do **not** push directly to main. Always branch + PR.
- Each week's work must be in its own subfolder:
  studentLabs/<REG>/lab01/
  studentLabs/<REG>/lab02/
  ...

Quick Reference

```
# Clone
git clone https://github.com/ORG_OR_USER/GamesENGII.git
cd GamesENGII
REG="SETU202212"

# New lab week
git checkout main
git pull --rebase
LAB="lab01"
git checkout -b "${LAB}/${REG}"
mkdir -p studentLabs/$REG/$LAB
echo "int main(){return 0;}" > studentLabs/$REG/$LAB/main.cpp
git add studentLabs/$REG/$LAB
git commit -m "${LAB}: submission for $REG"
git push -u origin "${LAB}/${REG}"
```

Then open a PR on GitHub.

Final Notes

- Use branches and Pull Requests responsibly.
- Keep commits clean and meaningful.
- Your folder = your responsibility.

Happy coding 🎮

Contact

Availability: **C202** (Wednesday 1500 - 1700 hrs)

For more info/help , email me at omer.ali@setu.ie