

# Wissenschaftliches Programmieren für Ingenieure

T. Achkar, Dr. D. Weygand, Dr. M. Stricker

## Übung 3 ( 2 Termine)

### Game of Life

### Eclipse : Entwicklungsumgebung, Makefile, Eigen3

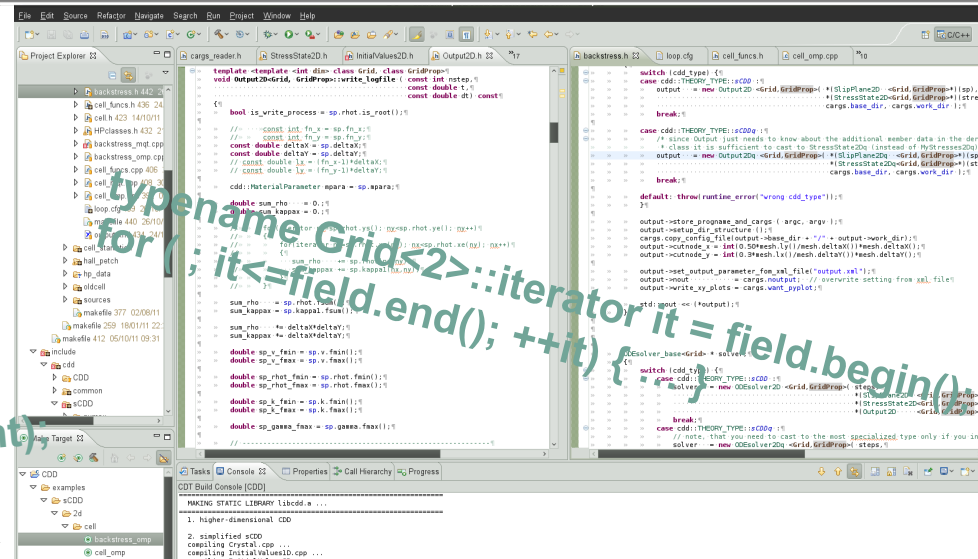
Institute for Applied Materials – Computational Materials Science (IAM-CMS)

KIT – The Karlsruhe Institute of Technology – University of the State of Baden-Wuerttemberg, GERMANY

class MyClass

```
{
public:
    MyClass() {}
};
```

(\*this)(i,j) = A.get\_coord(my\_point);



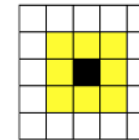
# Prinzip „Game of Life“

## Motivation: Game of Life

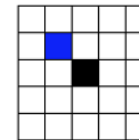
### Info

- Computersimulation des Lebens
- John Conway 1970
- Nachbarschaft bestimmt ob Zellen leben oder sterben (siehe rechts)
- ähnliche Modelle werden verwendet um Dendritenwachstum, Strömungen, Rekristallisation oder Verkehrsaufkommen zu simulieren

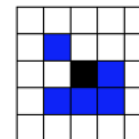
### Rules for the Cellular Automation Game 'Life'



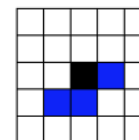
Each cell has 8 neighbors



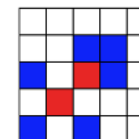
if an occupied cell has 0 or 1 neighbors, it dies (loneliness)



if an occupied cell has 4 to 8 neighbors, it dies (overcrowding)



if an occupied cell has 2 or 3 neighbors, it survives to the next generation



if an unoccupied cell has 3 occupied neighbors, it becomes occupied (birth)

# Gliederung der Übung

## Info

Die Übung soll in der Programmieroberfläche **Eclipse** bearbeitet werden.  
Die Übung besteht aus folgenden Teilaufgaben:

- Aufgabe 1: Bedienung von Eclipse, Compilieren und Ausführen eines **Makefile-Projektes**
- Aufgabe 2: Benutzung der <**Eigen3**> Bibliothek in einem Eclipse Projekt
- Aufgabe 3: Programmierung von **Game of Life**

# Aufgabe 1: Makefile Project

Todo

Benutzung einer integrierten Entwicklungsumgebung: eclipse

Laden Sie von ILIAS 'MakefileBeispiel.tgz' herunter und entpacken Sie es im Ordner Uebung3\_4:

**tar -zxvf MakefileBeispiel.tgz**

**Ziel:** Sie sollen das Programm, bestehend aus mehrere Teilen in eclipse editieren (verändern) und übersetzen können: die Sammlung der Dateien wird im Folgenden als Projekt bezeichnet. (Tutorial siehe Folie 13)

- Öffnen Sie Eclipse mittels Konsole (**eclipse &**) (Version 3.8.1).  
Nehmen Sie die default Einstellungen für den Workspace: die ist Ihr „Arbeitsplatz“, um das Projekt zu bearbeiten und zu übersetzen
- Erstellen Sie ein Makefile Projekt mittels: **File**→**New**→**Project**→**C/C++**→**Makefile Project with Existing Code**.
- Wählen sie einen „Project Name“: **Addition**
- Existing Code Location: **Pfad von 'MakefileBeispiel'**(Ordner Uebung3\_4 mit voller Pfadangabe)
- Toolchain for Indexer Settings: **Linux GCC** (compiler) (hier wird festgelegt welcher Compilersuite sich um das Projekt kümmern soll).

# Aufgabe 1: Makefile Project

Todo

Ziel: Makefile verstehen

Öffnen Sie die Dateien durch Anklicken im „Project Explorer“ (auf der linken Seite)

- **Lesen** und **verstehen** Sie das Makefile und die übrigen Dateien (\*.cxx, \*.h)  
(Vergleich Vorlesungsfolien: Namen der Dateien unterscheiden sich.... )
  - Warum verwendet man ein Makefile?
  - Welche Vorteile hat ein Makefile?
  - Was bewirkt die Präprozessoranweisung in addiere.h?

# Aufgabe 1: Makefile Project

Todo

Projekt compilieren und ausführen

Sie können nun das Projekt erstellen, d.h die ausführbare Datei (Program) erzeugen. Das Program können Sie sowohl direkt innerhalb von eclipse ausführen als auch in der Kommandozeile:

## ■ ECLIPSE

- Compilieren Sie mit Hilfe des Makefiles: **Project**→**Build Project**
- **Toolbar** Rechtsklick-→**Customize Perspective**-→**Command Groups Availability**: Option **Launch** auswählen
- Führen Sie das Programm aus: **Run** → **Run Configurations** → **C/C++ Application**: '**Programm Name**'; Danach **Run**→**Run**
- Zum Löschen der Objektdaten und Programm: **Project**→**Clean**

## ■ **Konsole**: Gehen Sie in der Konsole in den Projektordner:

- Compilieren Sie, das Projekt mittels dem Befehl:
- **make** und
- führen das Programm aus
- Löschen der Objektdaten und des Programms: **make clean**

# Aufgabe 2: <Eigen3> Bibliothek

## Todo

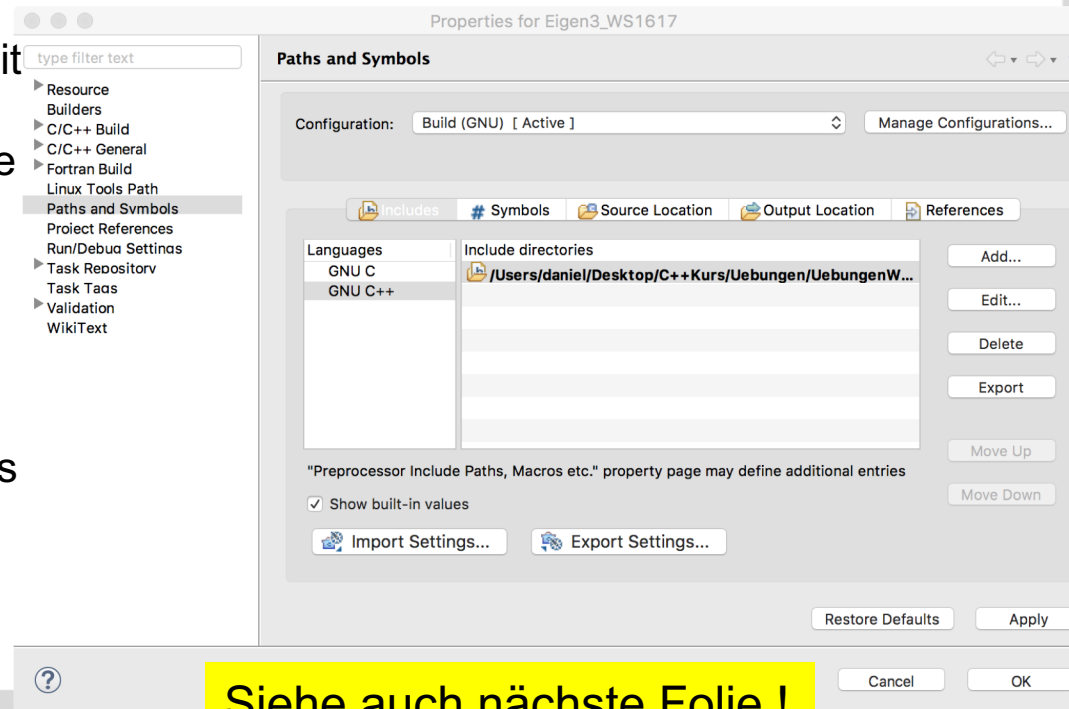
## Verwendung der Bibliothek für Vector Matrix Operationen

- Laden Sie von ILIAS eigen3.tar.gz' herunter und entpacken Sie es im Ordner Documents:  
**tar -zxvf eigen3.tar.gz**

Im folgenden wird die Bibliothek anhand von Beispielen getestet, dazu:

- Laden Sie von ILIAS test\_eigen3.tar.gz' herunter und entpacken Sie es im Ordner Uebung3\_4.
- unter „Anlegen von C++ Projekten mit bestehendem Makefile“ öffnen Sie das Projekt in eclipse und passen Sie den Pfad zur Bibliothek im Makefile an
- Einfügen in Eclipse zur Aktivierung der automatischen Ergänzung/Hilfe für Eigen3: Rechtsklick auf **Projekt ->Path and Symbols** : bei Languages GNU C++

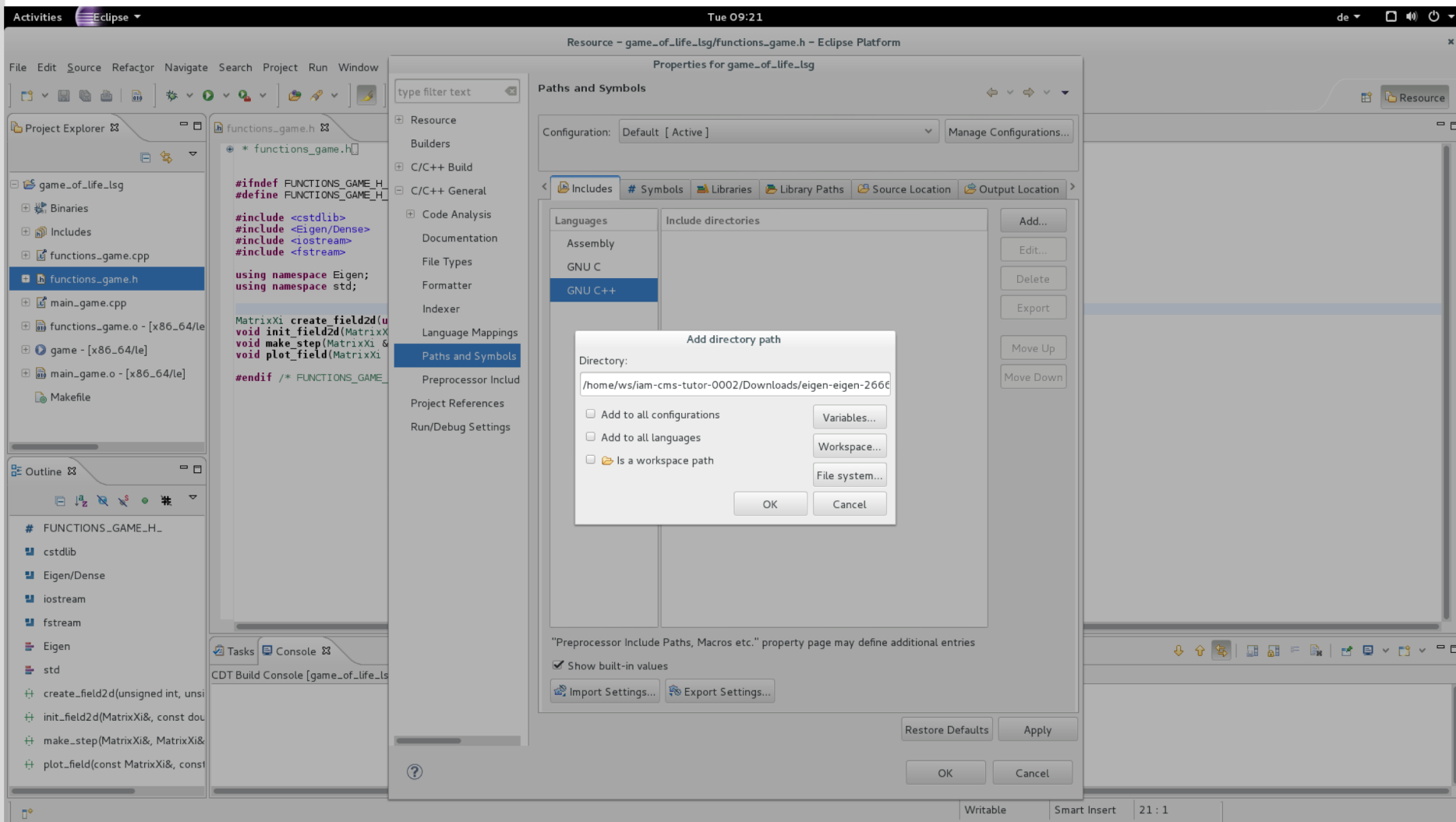
Add : included Directory und dort Pfad zu Ihrer entpackten eigen3 Biobliothek angeben (im SCC: Paths and Symbols unter C/C++ General)



**Siehe auch nächste Folie !**

# Eclipse 3.8.1 SCC Linux

Rechtsklick: Properties und dort Pfad eingeben





## Aufgabe 2: <Eigen3> Bibliothek

Todo

Verwendung der Bibliothek für Vector Matrix Operationen

- In der Datei test\_eigen.cxx erstellen Sie:
  - Vektoren und Matrizen auf verschiedenen Arten
  - Funktionen mit Matrix/Vector als Übergabeparameter (Beachte const und Referenz)
  - Verschieden Operationen, wie Skalarprodukte, Kreuzprodukte, usw.

(Siehe Vorlesungsfolien und <http://eigen.tuxfamily.org/dox/>)

Makefile anpassen: Pfad zur Bibliothek setzen, so dass er Ihrer Installation entspricht

CFLAGS=-O2 -I /home/ws/IHR\_NAME/IHR\_PFAD/eigen3/  
#hier muss der Pfad angepasst werden

- -I PFAD  
dahinter steht ein Pfad, in welchem weitere Headerdateien (hier die von Eigen3) zu finden sind

# Aufgabe 3: Game of Life

## Todo

- Laden Sie von ILIAS `game_of_life_vorlage.tar.gz` herunter und entpacken Sie es im Ordner `Uebung3_4`
- In Eclipse öffnen und erstellen Sie ein neues: **Makefile with existing Code**

## Info

## Projektaufbau

- Das Projekt besteht aus folgenden Dateien:
  - **main\_game.cpp** (das Hauptprogramm)
  - **functions\_game.cpp** (die zu implementierenden Funktionen)
  - **functions\_game.h** (die entsprechende Header Datei)
  - **makefile** (Übersetzungsvorschriften und Abhängigkeiten)

Es handelt sich um ein Programmgerüst: ergänzen Sie die fehlenden Codeteile -> Beschreibung auf den folgenden Folien !!

# Aufgabe 3: Game of Life

Todo

Implementierung


Schreiben Sie das Programm „Game of Life“ mit den Regeln aus Folie 2:

- Folgende Parameter sollen eingelesen werden:
  - Größe des 2 dimensional Feldes: **nZeile**, **nSpalte**
  - Anteil der besetzten Felder: **frac** (z.B. 0.5 bedeutet zu 50% besetzt)
  - Zahl der Zeitschritte: **nstep**
  - Ausgabe jeden n-ten Schritt: **nplot**
- Im **Hauptprogramm** werden:
  - Die Felder angelegt (`create_field2d`)
  - Und zufällig mit 0 und 1 initialisiert (`init_field2d`)
  - Über eine Zeitschleife die Bedingungen aus Folie 2 angewandt (`make_step`)
  - Das Feld in eine Datei geschrieben (`plot_field`)
- Mit **`create_field(...)`** ein zweidimensionales Feld (Größe `nx,ny`) anlegen und die Werte auf 0 setzen (Siehe die Funktion `Zero()` bei <http://eigen.tuxfamily.org/dox/>)

# Aufgabe 3: Game of Life

Todo

Implementierung

- Mit **init\_field(...)** die Zellen füllen
  - Verteilen Sie **zufällig** (besetzt) solange bis der Anteil der besetzten Zellen dem von Ihnen vorgegebenen Wert **frac** entspricht.  
**Hinweis:** Nutzen Sie den **Modulo** Operator % und den Zufalls generator **rand()**.  
Z.B.  $\text{rand()} \% nx \rightarrow$  zufällige Zahl zwischen 0 und  $nx-1$  
- Wenden Sie in **make\_step(...)** die Regeln aus Folie 2
  - Diese Funktion wird im Hauptprogramm für jeden Zeitschritt aufgerufen
  - Werten Sie je Zelle 8 Nachbarn aus. Verwenden Sie dafür **periodische Randbedingungen**
    - **Hinweis:** nutzen sie Modulo für Periodizität
    - z.B:  $(i-1+nx) \% nx \rightarrow$  Werte zwischen 0 und  $nx-1$
  - Nutzen Sie ein temporäres Feld zum Zwischenspeichern der aktualisierten Werte
  - Aktualisieren Sie zum Schluss das alte Feld

# Aufgabe 3: Game of Life

Todo

Visualisierung

- Die Funktion **plot\_field(...)** schreibt zu jedem Ausgabeschritt die Datei `field'step'.dat` mit dem Feld
- Verwenden Sie **Gnuplot**, um die Dateien zu visualisieren:
  - Öffnen Sie Gnuplot in der Shell im Ordner `game_of_life_vorlage`
  - Passen Sie den Plotbereich an:
    - `set xrange [0:nx]`
    - `set yrange [0:ny]`
  - Um über alle Dateien zu iterieren, verwenden Sie eine Schleife:
    - `do for [ii=0:nmax:delta]{file=sprintf('field%d.dat',ii);plot file u 1:2:3 with image}`  
oder
    - `do for [ii=0:nmax:delta]{file=sprintf('field%d.dat',ii);plot file u 1:2:3 with image;pause -1}`: damit kann man mit Return den nächsten Zeitschritt sehen.



# Aufgabe 3: Game of Life

## Zusatz

- Systemgröße: 100x100 oder mehr
- Anfangsdichte der Population variieren
  - Berechnen Sie zu jedem Zeitschritt die Dichte der Population.
  - Was beobachten Sie für geringe Anfangsdichten ?

# Tutorials für die Programmieroberfläche

## Eclipse CDT für C++

### Zusatz Info

- Eclipse beinhaltet eine Reihe von Tutorials zur Einführung in die IDE !!
- Diese Tutorials können Sie zur Vertiefung verwenden!

