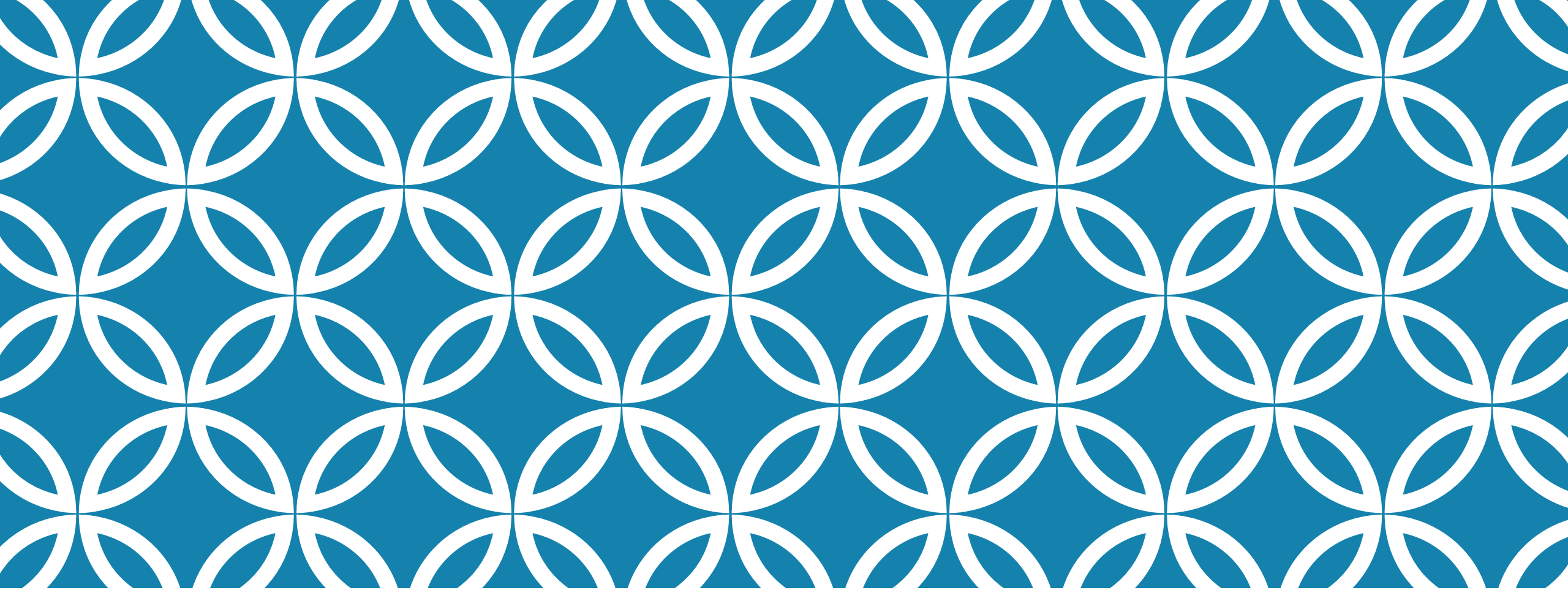


ASSIGNMENT #12: FINAL PROJECT MAKING A ROLE-PLAYING GAME

Introduction to Computers II

REVIEW ON PREVIOUS WORKS

Assignment #7 (Inheritance)

- `Players` with very simple skill system (`heal()` and `pray()`)

Assignment #8 (Polymorphism)

- `Monsters`
- `Players` and `Monsters` can attack each other
- Virtualize `specialSkill()` (combine `heal()` and `pray()`)
- Serialization feature

Assignment #9 (Serialization)

- Battle system (`Battle` class), multi-player versus multi-monster
- Money system

REVIEW ON PREVIOUS WORKS

Assignment #10 (Trying to write test files)

- Map system (`Field` class)
- Players will encounter monsters then enter (form) a battle
- A main procedure (`main.cpp`) with simple storyline

Assignment #11 (Run-time type checking/casting)

- Item system (`Item` series classes)
- Players can view their “backpack”
- Players can change their equipment such as weapons and armors
- Players can use consumable items (while they are in or not in a battle)
- Monsters will drop money and items after they dead

TASKS

1. **Implement game-save feature**
2. **Design and Implement the following classes**
   - `Game` class
   - `Menu` series classes
   - `Event` series classes
3. **Make a simple Role-Playing Game (RPG)**
   - Combine all your works so far (HW#7, #8, #9, #10, #11, #12)
   - Complete a full game with full features and story
   - (see “Final Project Requirements” page)

GAME-SAVE

Save the current game status to a file (SAVE)

- Player list
- Players' status (HP, MP, level, equipment, inventory, …)
- Players' current position
- Storyline progress (which events are completed/triggered?)
- Event list
- …

Your game should able to load status from a file as well (LOAD)

You can implement this using serialization feature we've done before

MAKE THE ARCHITECTURE MORE CLEAR

After studying chapter 25 and 26, we'd like to refactor our code with a more clear and descriptive architecture

We now introducing the following classes:

- `Game` class
- `Menu` class
- `Event` class

For the concept in this three new classes (and object-oriented design), please refer to chapter 25 and 26 (ATM case study)

Game CLASS: CONCEPT

This class wraps all top-level procedures of your game

- Control flow
- Game logics
- Game status
- …

Reference: the ATM class in chapter 26 of the textbook

Game CLASS: DATA MEMBERS

The data members are current game status:

- Player list
- Current position and map
- Storyline progress
- Files loaded
- Statistics (optional, e.g., # of monster killed)
- …

Game CLASS: SAMPLE main.cpp

```
// A super concise main.cpp
#include “Game.h”
using namespace std;

int main(void) {
    Game myRPG;
    myRPG.run();
    return 0;
}
```

Menu CLASS: CONCEPT

This class wraps the menu operations

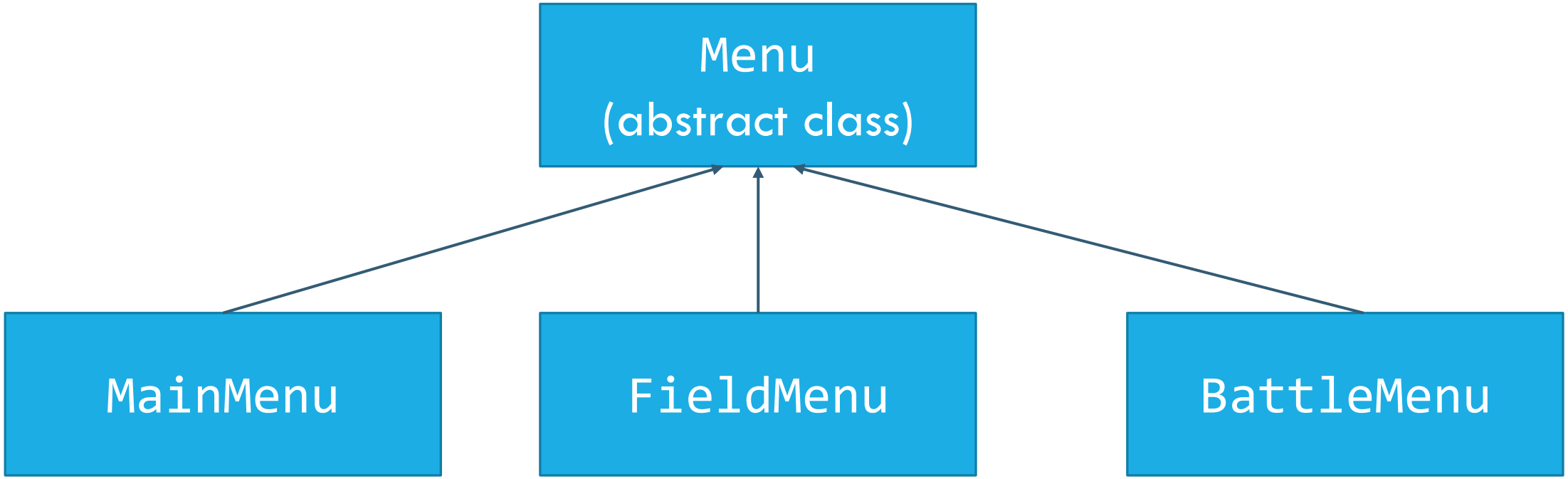
This class is responsible for:

1. Display the menu
2. Handle user input
3. Do/Control the actions according to user's input

You also can use this class as an UI handler, for example:

- User's inventory UI (drop/use items, view backpack, etc.)
- User's equipment UI (change equipment)
- Store UI (buy/sell items)

Menu CLASS: SAMPLE INHERITANCE HIERARCHY



Event CLASS: CONCEPT

Events play a critical role in **storyline progress**, an event can be:

- A dialogue/story
- Part of a task
- A battle
- Something found (chest, trap, etc.)
- …

By triggering these events within a game, the main storyline will move

We even can say that a game is composed of many events

Event CLASS: CONCEPT

An event may have one or more prerequisites of the following:

- Level limit
- Job limit
- Never being triggered (i.e., this event only can be triggered once)
- A particular event is completed (i.e., you need to complete another event first)

Some of events can be triggered many times

- E.g., branch tasks, instance dungeon tasks (副本任務) and daily tasks

Event CLASS: EXAMPLE

An event is like (*condition/action to trigger*, *content*):

- *Talk to the navigator* of game, he/she will *ask you to beat a monster*
- *After you completing this task*, the navigator will *give you a weapon*
- *Having a dialogue of villagers*, they will *give you some guide*
- There may be several *branch tasks (not necessary to do) with good rewards*
- When you *walk to a particular place*, you *find a chest with great treasure*
- Then after you *leveling-up to Lv.20*, you are *able to challenge the boss*
- You need to *talk to a gatekeeper (NPC)*, then *a battle with boss starts*
- After *you beating the boss*, there *comes the ending of the game*…

WRAP-UP: A SIMPLE EXAMPLE OF GAME FLOW

1. Your program should create an object of `Game` first
2. Then an object of `MainMenu` is created, waiting for user's input
3. After game starting, an object of `Field` is created
4. This instance of `Field` will load a map then display it to user
5. Then an object of `FieldMenu` is formed, waiting for user's command
6. Users may encounter monsters (entering a `Battle`) or triggers an `Event`
7. If a `Battle` is formed, display `BattleMenu`, handling user's action at each turn

FINAL PROJECT REQUIREMENTS

1. Your game should include ALL features we've done before (HW#**7**~**12**)
2. Your game should include:
   - At least 4 kinds of jobs
   - At least 4 kinds of monsters
   - At least 3 `Fields`
   - At least 6 `Items` (2 for each type)
   - At least 2 `Events`
   - At least 2 players within your team
3. A clear storyline (from beginning to the end)
4. Implement features beyond requirements will get bonus credits!

BEING CREATIVE!

You can modify/add any features of your game FREELY (but not necessary)

- Change formulas and predefine values
- More characters, monsters, items...
- More systems, for example:
  - **Character attributes** (strength, dexterity, poison resistance...)
  - **Stores**: buying potions and weapons, etc.
  - **Items refining** (物品精煉): makes your weapon/armor more powerful
  - **Alchemy** (物品合成/鍊金術): Item A + Item B = Rare item C
  - **A more complex skill system**: Skill tree, etc.
  - **Job-changing**: `NovicePlayer` to `KnightPlayer`
  - **Hit-rate**: Players'/Monsters' attack may failed (miss)
  - …

DELIVERABLES

1. **ALL of your source codes**
   - All class implementations and headers
   - `main.cpp` (or more files according to your implementation)
2. **A report (document) of your game**
   - (Please read the next page)
3. **A pre-compiled executable file (*.exe in Windows)**
   - Platform is not limited, but Windows (64-bit) is recommended
   - If you compiled your code on different platforms, please contact TAs

Please compress them into a zip archive then upload to Moodle

REPORT

Your report should include:

1. **How to play**
2. **Several screenshots of your game**
3. **How you implement each system/feature**
   - E.g., You stored items with a `vector` of a `struct`, the details are…
4. **Additional features beyond requirements you've done (if any)**
   - E.g., mutual restriction of five-phases (五行相剋)
5. **Third-party libraries you've used (if any)** (e.g., wxWidgets)
6. **UML diagrams we've introduced in chapter 25 (bonus, optional)**