



Homework 5

Mini Monopoly

Instructor: Tsung-Che Chiang

tcchiang@ieee.org

Department of Computer Science and Information Engineering

National Taiwan Normal University

Task

- Form a team with 4 members.
- Write a program that functions identically to (or even better than) the demo program on moodle.
- Use what you have learnt in our class to make your program correct, clean, clear, and extensible.

**Let's play the game for
5-10 minutes.**

Download *RichMini.exe* and *map.dat* from moodle.

To Write the Program ...

- Think about
 - What is the program flow?
 - What classes do we need?
 - What is the relationship between classes?
 - What are the data members and where should we store them?
 - What are the member functions?

To Complete the Task ...

- Think about
 - How does the team work together?
 - What is each member's job?
 - What are the milestones and how much time is allocated for each?

What classes do we have?

- At the first glance, we find that we may need:

- Player

- Land

- Map

- Game

```
= = [0]      USA <0> U$  400 L1  = = [9]  Russian   B$ 2000
= = [1]      Norway <2> U$  300 L1  = = [8]   Brazil <0> x1
= = [2]      Denmark  B$ 4000  = = [7]   Taiwan   B$ 2000
= = [3]      Germany <1> U$  200 L1  = = [6]    China <1> x1
=012= [4]     Poland <0> U$  800 L2  = = [5]    Spain <2> U$  600 L1

  [0]          A-Tu   $18400 with 3 units
  [1]        Little-Mei $24600 with 2 units
=>[2]         King-Baby $20200 with 2 units

King-Baby, your action? <1:Dice [default] / 2:Exit>...>
```

You may have your own design.

What classes do we have?

- After playing for 5-10 minutes, you know more about the game.

Who is here

```
= = [0]      USA <0> U$  400 L1    = = [9]      Russian      B$ 2000
= = [1]      Norway <2> U$   300 L1  = = [8]      Brazil <0>  x1
= = [2]      Denmark      B$ 4000    = = [7]      Taiwan      B$ 2000
= = [3]      Germany <1> U$   200 L1  = = [6]      China <1>  x1
=012= [4]     Poland <0> U$   800 L2  = = [5]      Spain <2>  U$   600 L1

[0]      A-Tu    $18400 with 3 units
[1]      Little-Mei $24600 with 2 units
=>[2]     King-Baby $20200 with 2 units

King-Baby, your action? <1:Dice [default] / 2:Exit>...
```

Player status: [ID] Name \$Money with # units

What classes do we have?

- After playing for 5-10 minutes, you know more about the game.

The screenshot shows a text-based game interface for Mini Monopoly. It lists various locations and their associated costs and levels. Annotations highlight specific features: 'Buyable' points to the cost field; 'Owner' points to the player name; 'Upgradable' points to the cost field; 'Level' points to the level field; and a note about Taiwan's random fines points to the question mark in the Taiwan entry.

Index	Location	Owner	Cost	Level
[0]	USA	[0]	U\$ 400	L1
[1]	Norway	[2]	U\$ 300	L1
[2]	Denmark		B\$ 4000	
[3]	Germany	[1]	U\$ 200	L1
[4]	Poland	[0]	U\$ 800	L2
[9]	Russian	[0]	x2	
[8]	Brazil	[0]	x2	
[7]	Taiwan	[2]	?	
[6]	China	[1]	x1	
[5]	Spain	[2]	U\$ 600	L1

Index	Player	Score	Units
[0]	A-Tu	\$16400	4 units
[1]	Little-Mei	\$24600	2 units
[2]	King-Baby	\$18200	3 units

King-Baby, your action? <1:Dice [default] / 2:Exit>...

Annotations:

- Buyable:** Points to the cost field (e.g., U\$ 400).
- Owner:** Points to the player name (e.g., A-Tu).
- Upgradable:** Points to the cost field (e.g., U\$ 300).
- Level:** Points to the level field (e.g., L1).
- Taiwan is a place with random fines:** Points to the question mark in the Taiwan entry.

What classes do we have?

- After playing for 5-10 minutes, you know more about the game.

```
= 2= [0]      USA <0> U$ 400 L1      = = [9]      Russian <0> x2
= = [1]      Norway <2> U$ 300 L1    = 1 = [8]      Brazil <0> x2
= = [2]      Denmark      B$ 4000    = = [7]      Taiwan <2> ?
= = [3]      Germany <1> U$ 200 L1   = = [6]      China <1> x1
=0 = [4]      Poland <0> U$ 800 L2   = = [5]      Spain <2> U$ 600 L1

=>[0]      A-Tu  $17400 with 4 units
    [1]      Little-Mei  $24400 with 2 units
    [2]      King-Baby  $17400 with 3 units

A-Tu, do you want to upgrade Poland? <1: Yes [default] / 2: No> ...>
You pay $800 to upgrade Poland to Lv.3
請按任意鍵繼續 . . .
```

What are the data members?

- For example, `Player` may have the following data members:

- `id_`
- `name_`
- `location_`
- `money_`
- `num_units_`

Note.

Data members listed here are only for your reference. You may have other choices if you feel that they are better.

- Make the data members

- `const` (not modifiable after initialization)
- `constexpr` (compile-time constants)
- `static` (shared global object within a class)

if necessary and proper.

What are the data members?

- Follow the rules:

- “*Always Initialize Variables.*”
 - Use in-class initializes and define/call constructors properly.
- “*Avoid Magic Numbers.*”
- “*Minimize Global & Shared Data.*”
- “*Avoid Macros.*”
 - No `using` in header files.

What are the data members?

- Classes of players and their relationship

- `class Player`

- a class having the data members as mentioned.

- `class WorldPlayer`

- a class containing an array / vector of (pointers to) `Player`.

What are the data members?

- There are four types of **map units**:

- **Upgradable**

- The fine is based on the level.



Check *map.dat* file.

- **Collectable**

- Fine \$ = "the number of collectable units of the owner" × "unit fine".

- **RandomCost**

- Fine \$ = "dice points [1-6]" × "unit fine".

- **Jail**

- The player is frozen for one round.
 - Jail is not implemented in the demo program, but you need to implement it in your program.

What are the data members?

- For example, the `Upgradable` unit may have the following data members:

- `id_`
- `name_`
- `price_, upgrade_price_, travel_fine_`
- `level_`
- `host_`
- `who_is_here_`

Note.

Data members listed here are only for your reference. You may have other choices if you feel that they are better.

- The four kinds of units may have common data members as well as specific data members.

⇒ Put common ones in the base class.

What are the data members?

- To manipulate the units, we need a **WorldMap**, which is a collection of units.
- We cannot have one array/vector to store different kinds of objects directly.

⇒ Recall that a pointer of the base class can point to the objects of derived classes.

What are the data members?

■ Classes of units and their relationship

■ `class MapUnit`

- a base class for defining the common data members and common interfaces for the various kinds of map units

■ `class UpgradableUnit` (you may give it a better name)

- a class derived from `MapUnit` having the data fields as mentioned earlier

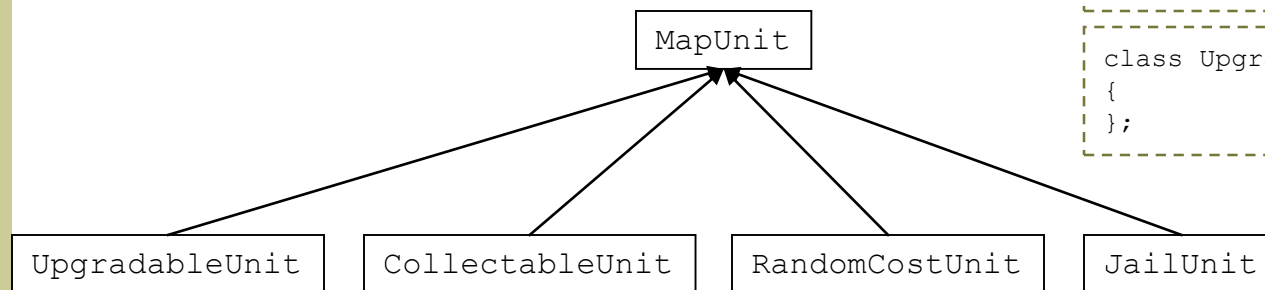
■ `classes CollectableUnit, RandomCostUnit, and JailUnit`

■ `class WorldMap`

- a class containing an array / vector of pointers to `MapUnit`.

What are the data members?

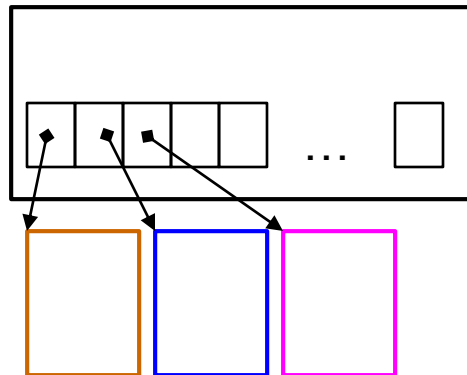
All four specific map units inherit from CMapUnit.



```
class MapUnit  
{  
};
```

```
class UpgradableUnit : public MapUnit  
{  
};
```


A WorldMap object contains (MapUnit) pointers to derived-class objects.



```
class WorldMap  
{  
  
private:  
    MapUnit *units_[20]={};  
};
```

You can define the maximum number of map units and implement by an array here.

What are their member functions?

- The most important member functions are the **constructor**(s) and the **destructor** (if needed).
 - First, check if a default constructor must be written by yourself.
 - Then, check if you need a parameterized constructor.
 - For the constructor with a single parameter, `explicit` is often added.
 - For the classes with pointers as data members, you usually need a destructor.
 - Don't forget the rules:
 - *"Prefer Initialization to Assignment."*
 - *"Copy and Destroy Consistently."*  To be explained in the next class.
 - ...

What are their member functions?

- For your classes, provide suitable `public` member functions to access and manipulate `private` data.
 - Sometimes, you may also write `protected` or `private` member functions just for the ease of implementation.
- Make your member functions `const` whenever you can, and make them `static` if they access only static data members.
- Overload operators (e.g. `<<`, `[]`, etc.) for convenience.

What are their member functions?

- We have several kinds of map units. How can we manipulate them in a uniform way, which helps us to add more kinds of map units in the future?
 - How can we model their behaviors in the same pattern?
- What are the common member functions?
 - Which ones have the same implementation for all kinds of units?
 - Which ones have a different implementation for each kind of unit?

What are their member functions?

- In the class hierarchy:
 - When all types of units have the same behavior, define a `non-virtual` member function in the base class.
 - When all types of units have the same action (perhaps in different ways), make a `virtual` member function.
 - If possible, provide a default implementation in the base class and call it in derived classes' versions properly.
 - Make it a pure `virtual` member function to enforce derived classes to override.
- Use the `override` keyword to check prototypes.
- Call base class's constructors properly.



To be explained in the next class.

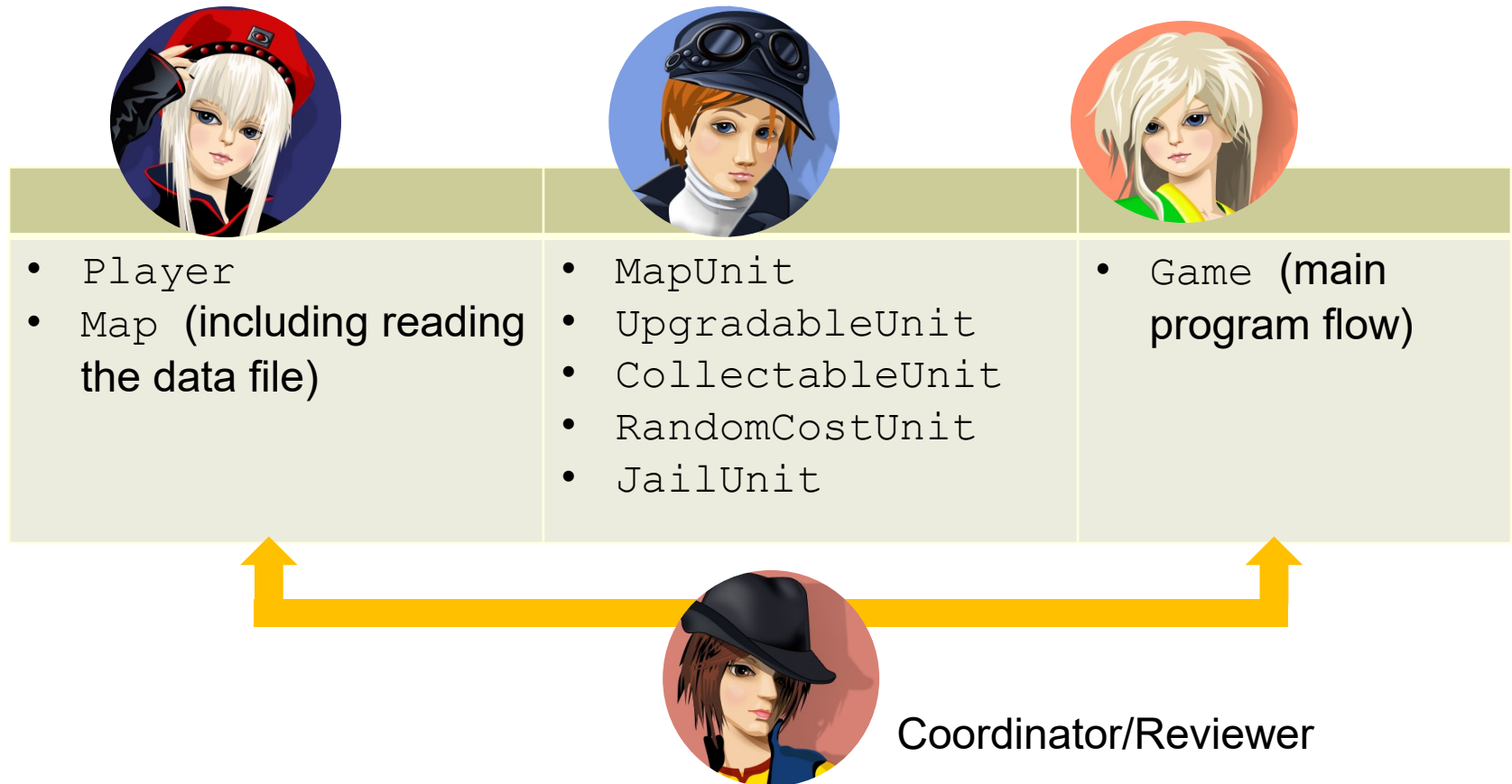
What are their member functions?

- Follow the rules:

- *“Avoid Long Functions & Deep Nesting.”*
- *“Declare Variables as Locally as Possible.”*
- *“Take Parameters Appropriately by Value, Pointer, or Reference.”*
- *“Use const Proactively.”*

Team Composition

Just a suggestion,
not necessary to follow





初始化

顯示

互動

剩一人?

Y

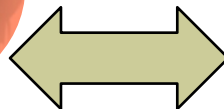
結束

N

MapUnit

CMapUnit.h
CMapUnit.cpp

RandomCostUnit



你們的物件怎麼初始化?
我要給你們什麼資料?
你們的物件有什麼功能?

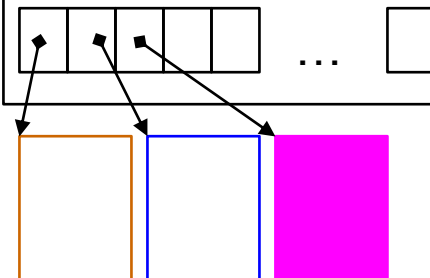
Player

WorldPlayer

WorldMap

WorldMap

CMapUnit *



Game Functions

- Run the demo program.
 - Note that some functions (Jail and round-trip reward) are not included yet.
 - If you find any bug, please report to me. Thanks!
- Key functions
 - Reading the map from a data file
 - The map is guaranteed to contain at least six units.
 - Setting number of players and their names
 - The minimum and maximum numbers of players are 1 and 4, respectively.
 - Dicing and moving
 - Each time when a player passes or stops at the starting point, he/she gets a fixed amount of reward.

Game Functions

- Key functions (continued)
 - When a player visits a buyable unit,
 - (s)he can **buy** it if nobody owns the unit and (s)he has enough money.
 - (s)he is **fin**ed if some other owns the unit. Different kinds of units have different ways of fining.
 - (s)he can **upgrade** it if (s)he owns the unit and the unit is upgradable. The maximum level is 5.
 - When a player visits a jail,
 - (s)he cannot move at the next round.

Game Functions

- Key functions (continued)
 - If a player does not have enough money (< 0), (s)he can not play the game any more.
 - Note that all the units owned by this bankrupt player should be released. Other live players can buy them.
 - The levels of upgradable units are reset to 1.
 - The game ends when only one player survives.
- Simulate the demo program as well as you can.

Format of Map File



[Back to map unit design](#)

U
Upgradable
C
Collectable
R
RandomCost
J
Jail

```
U USA 4000 500 400 800 1200 1600 2000
U Norway 3000 400 300 600 1000 1200 1500
U Denmark 4000 500 400 800 1200 1600 2000
C Italy 2000 100
U Poland 8000 900 800 2000 3500 4000 4500
C China 1000 100
R Taiwan 2000 500
J Jail
```

For **U**, **C**, and **R** units, the **first integer** after the name of the unit is the cost to buy the unit.

For **C** and **R** units, the *second integer* is the unit fine.

For **U** units, the *second integer* is the cost for each level-up.

The next integers are the fine of the units with level 1, 2, 3, 4, and 5, respectively.

Quick Check Items

- Define required data members at proper classes at proper access levels.
 - Be careful of their data types.
 - Do not define redundant data members.
- Define required member functions and their arguments properly.
 - Do not define redundant member functions.
 - Do not add unnecessary tasks to the member functions. (SRP)
 - Which functions should be `const`? Which should be `virtual`?
- What should be put in `.h` and what in `.cpp`?
- Be careful of dynamic memory allocation.

Submission

- **Deadline: 2019.6.24 0:00 (hard deadline)**
- One team submit one package in a whole. File name: **TPP2019-HW5-ooo-ooo-ooo-ooo.rar/.zip**. It includes:
 - all source files including the project file and map data file;
 - a **codes.pdf** file containing all of your codes with suitable comments in proper format to print out;
 - a **report.pdf** file, including:
 - a summary of everyone's task and contribution
 - every team member's 1~2 page report on what you learnt in this team project and this course.