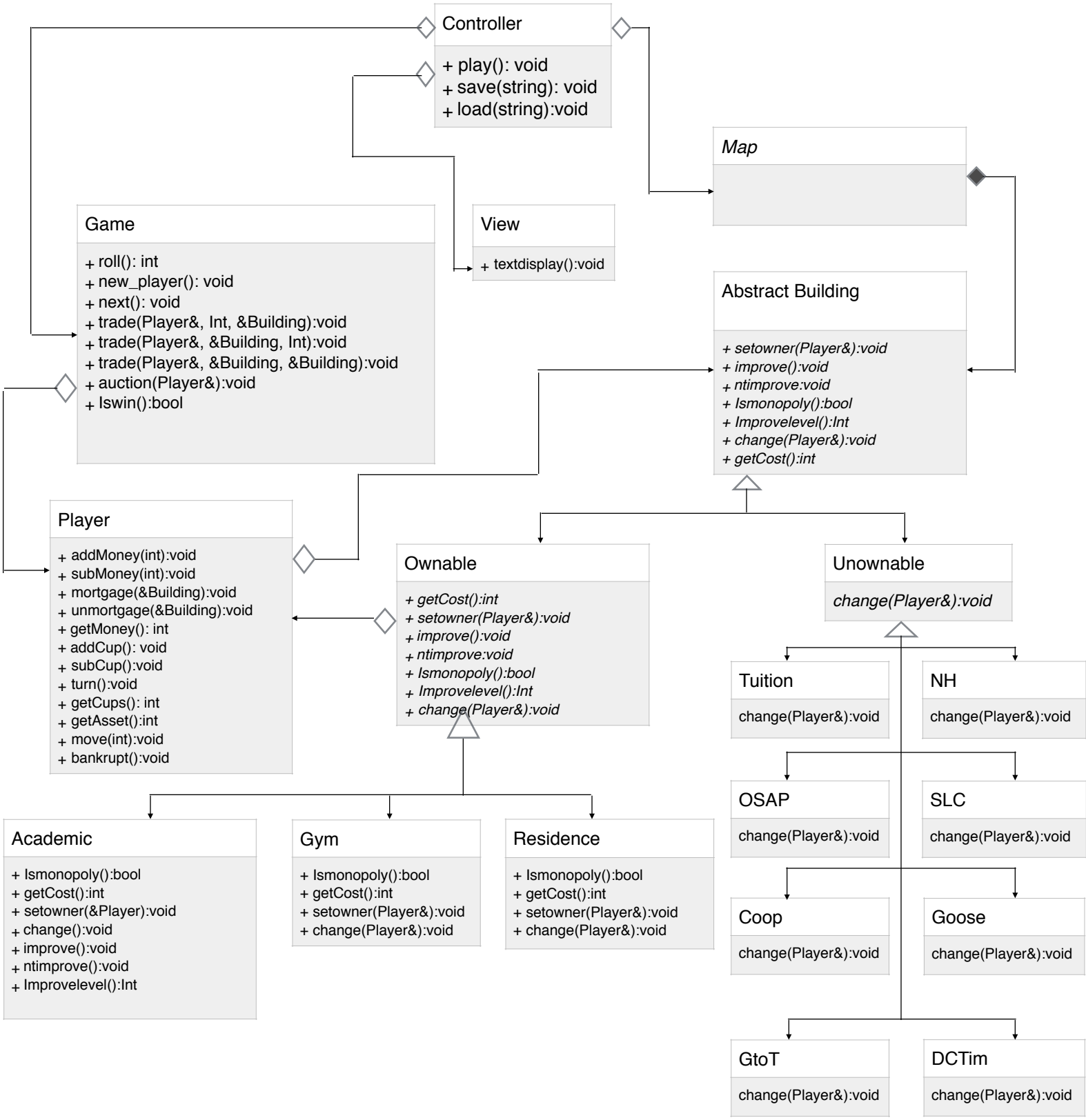
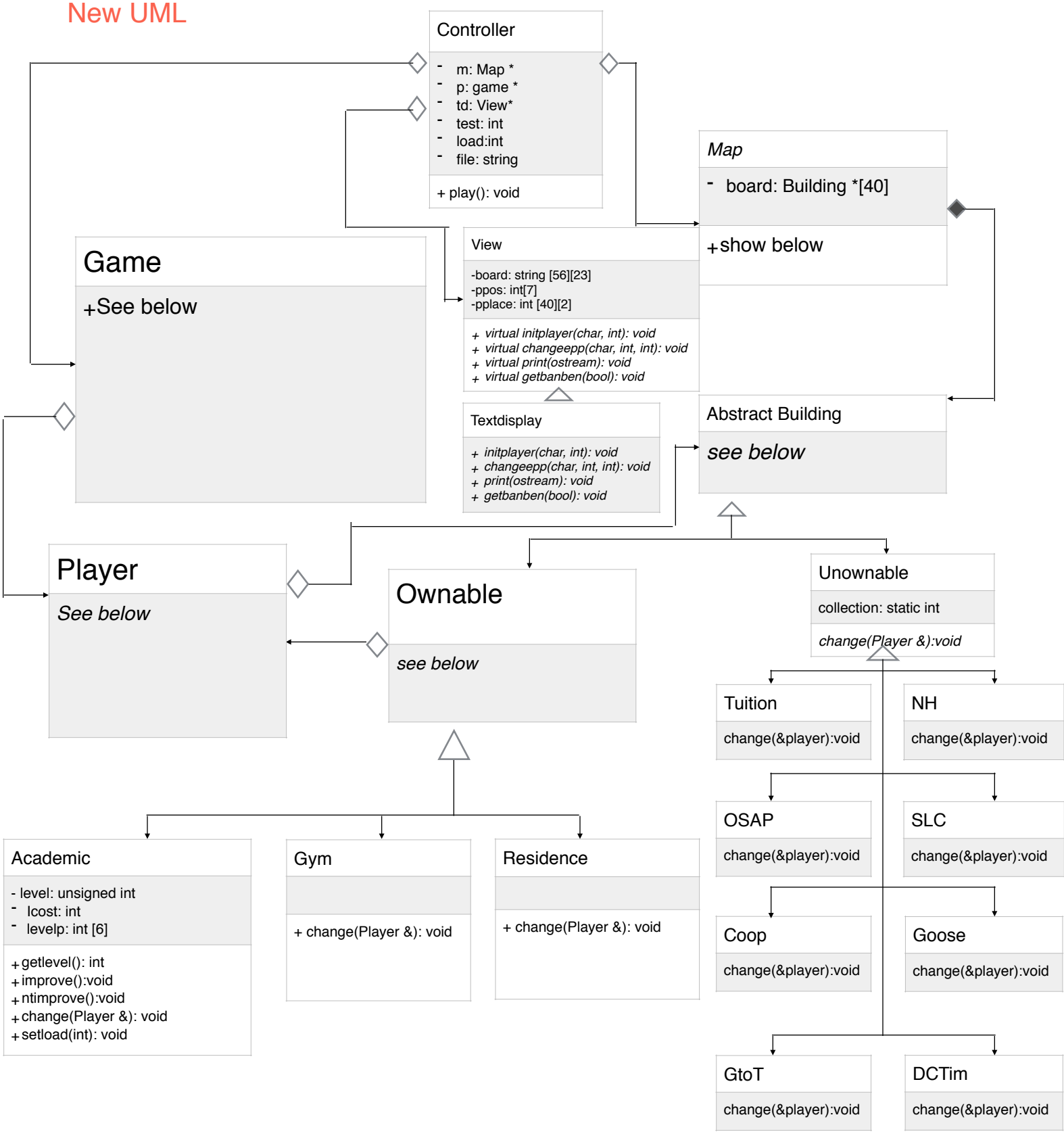
CS246 Final Project
BB7K

by:
Weixiang Deng
Weifeng Jiang

Old UML


Controller
+ play(): void
+ save(string): void
+ load(string):void
Map
Game
+ roll(): int
+ new_player(): void
+ next(): void
+ trade(Player&, Int, &Building):void
+ trade(Player&, &Building, Int):void
+ trade(Player&, &Building, &Building):void
+ auction(Player&):void
+ Iswin():bool
View
+ textdisplay():void
Abstract Building
+ setowner(Player&):void
+ improve():void
+ ntimprove:void
+ Ismonopoly():bool
+ Improvelevel():Int
+ change(Player&):void
+ getCost():int
Player
+ addMoney(int):void
+ subMoney(int):void
+ mortgage(&Building):void
+ unmortgage(&Building):void
+ getMoney(): int
+ addCup(): void
+ subCup():void
+ turn():void
+ getCups(): int
+ getAsset():int
+ move(int):void
+ bankrupt():void
Ownable
+ getCost():int
+ setowner(Player&):void
+ improve():void
+ ntimprove:void
+ Ismonopoly():bool
+ Improvelevel():Int
+ change(Player&):void
Unownable
change(Player&):void
Tuition
change(Player&):void
NH
change(Player&):void
OSAP
change(Player&):void
SLC
change(Player&):void
Coop
change(Player&):void
Goose
change(Player&):void
GtoT
change(Player&):void
DCTim
change(Player&):void
Academic
+ Ismonopoly():bool
+ getCost():int
+ setowner(&Player):void
+ change():void
+ improve():void
+ ntimprove():void
+ Improvelevel():Int
Gym
+ Ismonopoly():bool
+ getCost():int
+ setowner(Player&):void
+ change(Player&):void
Residence
+ Ismonopoly():bool
+ getCost():int
+ setowner(Player&):void
+ change(Player&):void

New UML


Controller
- m: Map *
- p: game *
- td: View*
- test: int
- load:int
- file: string
+ play(): void
Map
- board: Building *[40]
+show below
Game
+See below
View
-board: string [56][23]
-ppos: int[7]
-pplace: int [40][2]
+ virtual initplayer(char, int): void
+ virtual changeepp(char, int, int): void
+ virtual print(ostream): void
+ virtual getbanben(bool): void
Textdisplay
+ initplayer(char, int): void
+ changeepp(char, int, int): void
+ print(ostream): void
+ getbanben(bool): void
Abstract Building
see below
Player
See below
Ownable
see below
Unownable
collection: static int
change(Player &):void
Tuition
change(&player):void
NH
change(&player):void
OSAP
change(&player):void
SLC
change(&player):void
Coop
change(&player):void
Goose
change(&player):void
GtoT
change(&player):void
DCTim
change(&player):void
Academic
- level: unsigned int
- Icost: int
- levelp: int [6]
+getlevel(): int
+improve():void
+ntimprove():void
+change(Player &): void
+setload(int): void
Gym
+ change(Player &): void
Residence
+ change(Player &): void

Player

- name: string
- symbol: char
- momey: int
- total_cup: static int
- cup: int
- place: int
- own_num: int
- Rnum: int
- Gnum: int
- own: int [32]
- die: bool
- lose: bool
- map: Map *
- send: bool
- test: bool
- outer: bool
- auc: bool
- DCcount: int

+ setmap(Map &): void
+ gettest(): bool
+ set_die(): void
+ set_lose(): void
+ print_name(): void
+ getname(): string
+ check_havebuilding(int): bool
+ check_lose(): bool
+ check_die(): bool
+ getchar(): char
+ add_money(int): void
+ sub_money(int): void
+ add_cup(): void
+ sub_cup(): void
+ getMoney(): int
+ getCups(): int
+ getAsset(): int
+ mortgage(int): void
+ unmortgage(int): void
+ add_building(int): void
+ sub_building(int): void
+ move(int):void
+ getTcup():int
+ getMap(): Map *
+ getLoc(): int
+ setsend(bool): void
+ getsend(): bool
+ print_ownbuilding(): void
+ settest(): void
+ getDC(): int
+ setDC(int): void
+ setplace(int): void
+ getout():bool
+ etout(bool): void
+ getauc(): bool
+ setauc(bool): void
+ getownnum(): int
+ getGnum(): int
+ getRnum(): int
+ makeownarr(map<int,int>): void

Game

- num_player: int
- players: Player * [8]
- now_play: int

+ roll(int): int
+ loop(): void
+ get_nowplay(): int
+ start_new(Map &): void
+ load_player(ifstream &, Map &): void
+ getassets(): void
+ getname(): string
+ getname(int): string
+ getchar(): char
+ getchar(int): char
+ getcup(int): int
+ getmoney(int): int
+ getmoney(): int
+ getposition(int): int
+ getDC(int): int
+ setDC(int, int): void
+ add_player(string,char,Map &): void
+ add_player(string,char,int,int,int,Map&): void
+ trade_money(string,string,int,int): void
+ trade_building(string,string,int,int): void
+ next(): void
+ lswin(): bool
+ print_numplayer(): void
+ getLoc(): int
+ get_numplayer(): int
+ reset_nowplay(): void
+ print_winer(): void
+ mortgage(int): void
+ unmortgage(int): void
+ getnump(): int
+ getnplay(): int
+ makearray(int): map<int, Player *>
+ getplayer(string): Player *
+ setbuilding(string,int): void
+ setlose(): void
+ getnplayer(): Player *
+ start_new(Map &, bool, int): void
+ start_new(Map &, int): void
+ getassets(bool): void
+ add_player(string,char,Map&,bool):void
+ check_havebuilding(int): bool
+ check_havebuilding(int,string): bool

Ownable

- mort: bool
- monopoly: bool
- owner: Player*
- block: string
- blockmen: Building[4]

+ *virtual getowner(): Player **
+ *virtual setowner(Player &) : void*
+ *virtual getCost(): int*
+ *virtual addCost(int): void*
+ *virtual subCost(int): void*
+ *virtual getPay(): int*
+ *virtual setPay(int): void*
+ *virtual getmort(): bool*
+ *virtual improve(): void*
+ *virtual ntimprove(): void*
+ *virtual Ismonopoly(): bool*
+ *virtual setmonopoly(bool): void*
+ *virtual setunmonopoly(): void*
+ *virtual getlevel(): int*
+ *virtual addmem(Building): void*
+ *virtual getblock(): string*
+ *virtual getblockcap(): string*
+ *virtual buy(Player &): void*
+ *virtual auction(int, int, char, int, int, map<int Player*>, Building*, map<int, int>):void*
+ *virtual change(Player&): void*
+ *virtual setload(int): void*
+ *virtual setblock(string): void*

Map

- board: Building *[40]

+ change(int, Player &): void
+ printname(int): string
+ getowner(int): Player *
+ getname(int): string
+ getCost(int): int
+ getmort(int): bool
+ setOwner(int, Player&): void
+ improve(int): void
+ ntimprove(int): void
+ Ismonopoly(int): bool
+ setmortgage(int): void
+ setunmortgage(int): void
+ getPay(int): int
+ getlevel(int): int
+ getLoc(int): int
+ getblock(int): string
+ getbuilding(int): Building *
+ auction(int, int, char, int, int, map<int Player*>, Building*, map<int, int>):void
+ setload(int, int): void
+ getbanben(bool): void
+ Isownable(int): bool

Abstract Building

- name:string
- loc: int
- banben: bool

+ *virtual getowner(): Player **
+ *virtual setowner(Player &) : void*
+ *virtual getCost(): int*
+ *virtual addCost(int): void*
+ *virtual subCost(int): void*
+ *virtual getPay(): int*
+ *virtual setPay(int): void*
+ *virtual getmort(): bool*
+ *virtual improve(): void*
+ *virtual ntimprove(): void*
+ *virtual Ismonopoly(): bool*
+ *virtual setmonopoly(bool): void*
+ *virtual setunmonopoly(): void*
+ *virtual getlevel(): int*
+ *virtual addmem(Building): void*
+ *virtual getloc(): int*
+ *virtual getblock(): string*
+ *virtual getblockcap(): string*
+ *virtual buy(Player &): void*
+ *virtual auction(int, int, char, int, int, map<int Player*>, Building*, map<int, int>):void*
+ *virtual change(Player&): void*
+ *virtual setload(int): void*
+ *virtual getbanban(): bool*
+ *virtual setbanban(): void*
+ *virtual setblock(string): void*
+ *virtual setname(string): void*

Class:

the main class include:

- Controller
- Game
- Player
- Map
- Building
- View
- Testdisplay

In the main function consume the argument for testing and load file purpose, the main process of the game play are in the class Controller, then the Controller create it will construct the class Map , Game and View.

-> **Map**: the class Map is make up with buildings, it store every building in this game and their functionality.

-> **Building**: Building is a superclass that represent every single building in this game.

-> **Game**: the class Game contain every player in this game and their information.

-> **Player**: Player is a class that store the information of each single player.

Game Flow

When running the game, the game flow is

Controller::play() -> Game::roll(int) -> Player::move(int)

the controller read the command from the client, like roll, improve..etc. and then Game and Map operate the change process.

Differences:

We add a child-class Textdisplay for class View. However, we haven't changed the whole structure of the UML. We adds so many functions on each class because we consider totally on due date 1. Therefore, we miss so many functions we need to use. Then we adds them on the final version of our project.

Q&A

Q1: After reading this subsection, would the Observer Pattern be a good pattern to use when implementing a gameboard? Why or why not?

Answer: No, actually, we haven't used it for our code. We can use pointers strictly to get data from Building for Map(gameboard).

Q2: Suppose that we wanted to model SLC and Needles Hall more closely to Chance and Community Chest cards. Is there a suitable design pattern you could use? How would you use it?

Answer: Yes, the template pattern is useful. We can use it where NH and SLC would inherit from the common superclass(Unownable) with a change() virtual method. The method change() in SLC and NH can override it by themselves.

Q3: What could you do to ensure there are never more than 4 Roll Up the Rim cups?

Answer: We can make a static integer that count the total number of cups that all players own, and check as soon as each player comes to NH or SLC.

Q4: Is the Decorator Pattern a good pattern to use when implementing Improvements? Why or why not?

Answer: Yes, Academic buildings in the map is still buildings after implementing improvements, they do not change. So the decorator pattern is suitable for this situation. So we create a Building class with Ownable subclass with Academic subclass.

Q5: What lessons did this project teach you about developing software in teams? If you worked alone, what lessons did you learn about writing large programs?

Answer: By working on this program we realize that developing communicating with the teammate is very important,
It's good to have a plan on how to distribute the workload.
It's easier to collaborate if you're open minded and willing to own up to your mistakes.

Q6: What would you have done differently if you had the chance to start over?

Answer: We will deliver our job for each class and we will communicating with each other more frequently, that can save lots of time to compare to our codes. Also we do lots of function override because lack of communication. We will make the code be more efficient if we can do more communication before writing new functions.